

Brad Traversy: Laravel From Scratch

1. Setting Up Tools & Dev Environment:

PHP Intelephense, Laravel Blade Snippets & Laravel Snippets extensions | Laravel herd | virtualbox win10 vm

2. Getting Started with Laravel Routing:

web.php routing file | adding GET & POST endpoints | CSRF protection | Middleware CSRF exceptions | passing params in URI | param constraints | global constraints | request object | request object query params | query, only, all, has, input keywords | Response Helper | Response Header Http Code | Response Header content type | Response Header download | Response Header set cookie | read cookie | Herd secure self signing tls

3. Views & Controllers:

create & display views | pass data to view | with method | compact method | blade templates & directives | @if, @else, @foreach, @forelse, @empty, @break, @continue | \$loop->index, \$loop->iteration, \$loop->remaining, \$loop->count, \$loop->even, \$loop->odd | create controllers | using artisan | controller methods | clean web.php routing file | params and requests in the controller | @csrf hidden token | Generate Resource Routes & Methods | type hinting in controllers | Layouts with Template Inheritance | Partial & @include directive

4. Components & Styling:

what are components? | components & slots | named slots | z-header, x-slot | install tailwindcss & other dependencies via NPM | Header Component & URL Helper | Conditional Classes | @php Directive | Component Attributes | Props | nav.link & button-link blades using props & attributes | mobile menu nav links | mobile menu toggle | hero component | top and bottom banners

5. PostgreSQL Database setup & Migrations:

Database options | Create Database & user | Configure Database connection | Migrations Overview & Commands | Creating Migrations

6. Models, Eloquent ORM, Factories & Seeders:

Intro to Models | Fetching Data | Eloquent ORM | Tinker | CRUD Operations | Model Binding | Single Job Listing | Create Job Listing | Input Validation & Errors | Job Schema Update Migration | Eloquent Relationships | Using Factories | Creating Factory & Faker Library | Foreign Key errors when seeding to MySQL/MariaDb databases

Brad Traversy: Laravel From Scratch

7. Pages, Presentation & Crud:

Job Page | Job Card Component | Homepage Jobs | Job Details Page | Create Job Page | Text Input Component | Other Input components | Finish Input Validation | Flash Messages & Alert Component | Alpine.JS & Alert Dismiss | Optional Job fields | file uploading | dd (die & dump) | symlinks | storage folder | Update Job Listing | Delete Job Listing

8. Authentication & Creating users:

Authentication Options | Laravel Breeze Setup | How Sessions Work & Session Helpers | Login & Register Control Routes | Register New user | Log in user | Logout and Auth Directive

9. Middleware, Authorisation & policies:

Middleware Overview | Protecting Routes (in controller & in routes\web.php with middleware | Guest Middleware | Test User Seeder | Add Current user to Listing | Policies & @can Directive | Policy Authorisation in Controller

10. Dashboard, Profile & Pagination:

Dashboard Controller & View | Dashboard user Job Listings | Profile Controller & Info Update | Profile Avatar upload | Show Avatar in Header | Simple Job Pagination | customising pagination view

11. Bookmark Job Listings:

Bookmark Migration & Relationships | Seeding Bookmarks | Get & Show Bookmarks | Bookmarking Jobs | Removing Bookmarks

12. Job Applicants & Resume Upload:

Aplicant Migration & Model | Aplicant Form Modal with Alpine.js | Fix Modal Blip with x-cloak | Aplicant Controller & Store Method | Show Applications to Owner | Delete Applicants | Prevent Multiple Applications

13. Job Search, Maps & Emails:

Search Component & Route | Search Functionality | Mapbox Setup | Hide Mapbox key with proxy endpoint | Send Emails with Mailers & Mialtrap | Sending Data in Emails | Email Attachments | Setup Emails For Production

14. Deploy using Laravel Forge:

Prepare & Push to GitHub | Laravel Forge Server & Site setup | Domain name Setup | SSL & launch Test

1. Setting Up Tools & Dev Environment

[1.1 Add some Vscode extensions](#)

[1.2 Install Laravel herd](#)

[1.3 Laravel Dashboard](#)

[1.4 Create new site in Herd Dashboard](#)

[1.5 Open herd site project files in VScode](#)

[1.6 Open Project in Browser](#)

1.1 Add Some VsCode extensions

Add the following code extensions



PHP IntelliSense

Ben Mewburn  intelephense.com |  15,221,399 |  (393) |  Sponsor

PHP code intelligence for Visual Studio Code

Disable Uninstall ✓ Auto Update 



Laravel Blade Snippets

Winnie Lin |  4,255,283 |  (38)

Laravel blade snippets and syntax highlight support

Disable Uninstall ✓ Auto Update 



Laravel Snippets

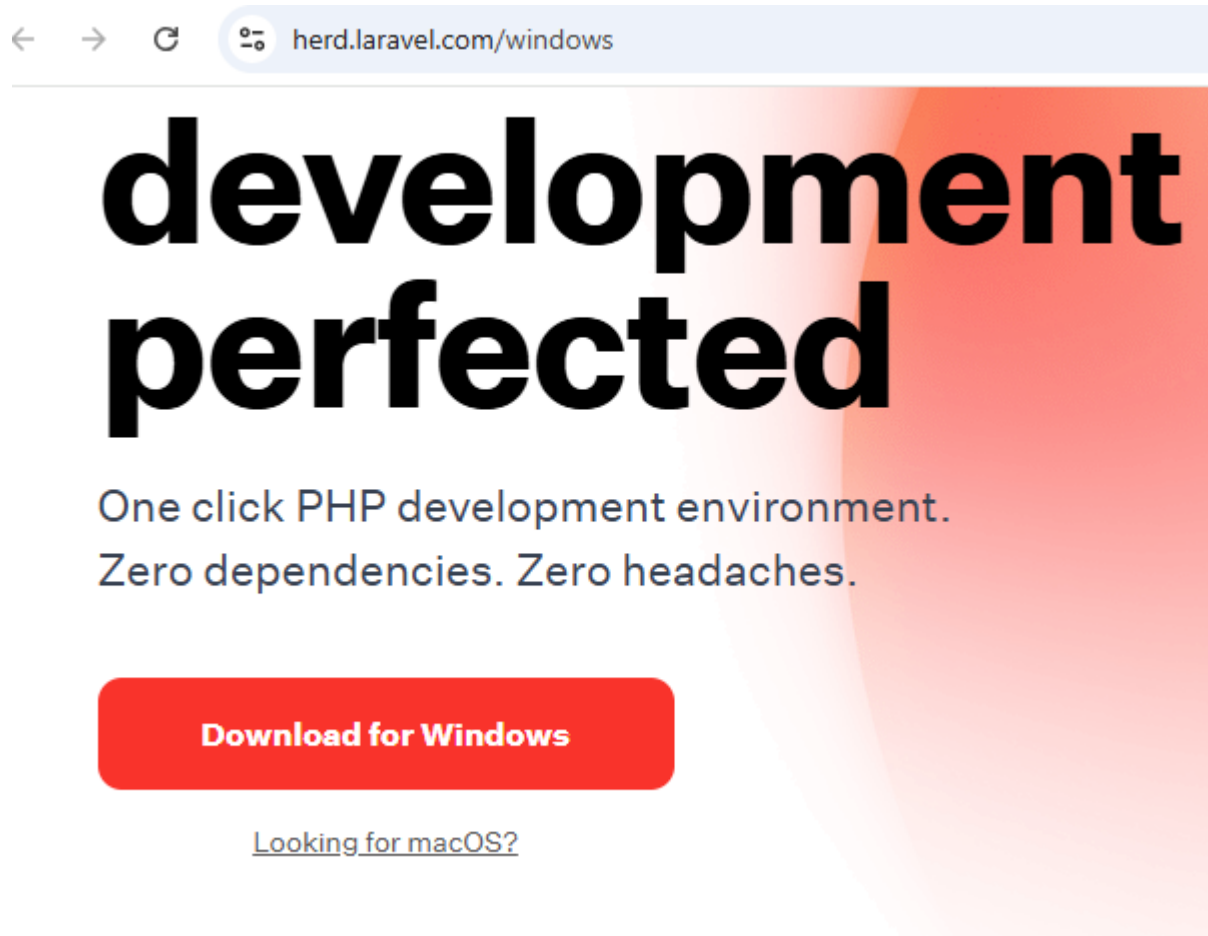
Winnie Lin |  2,820,548 |  (11)

Laravel snippets for Visual Studio Code (Support Laravel 5 and above)

Disable Uninstall ✓ Auto Update 

1.2 Install Laravel Herd

<https://herd.laravel.com/windows>



Download the Laravel herd setup executable. Follow onscreen prompts and allow this app to make changes to your device.

1.2 Laravel Dashboard

The screenshot displays the Laravel Herd dashboard interface. On the left is a sidebar with navigation links: Dashboard, General, Sites (highlighted), PHP, Node, Expose, Shortcuts, Services (Pro), Mail (Pro), Dumps (Pro), Herd Pro, Integrations, and About. The main area is titled 'Sites' and contains an 'Add' button, a search bar, and a list of sites under the 'Ungrouped' category. The 'workopia.test' site is selected, showing its configuration in the 'General' tab. This tab includes a preview window and a list of tools with 'Open' buttons: Terminal, PhpStorm, Tinker, and Adminer. Below these, the PHP version is set to 8.4, and the Path and URL are displayed.

Tool	Action
Terminal	Open
PhpStorm	Open
Tinker	Open
Adminer	Open

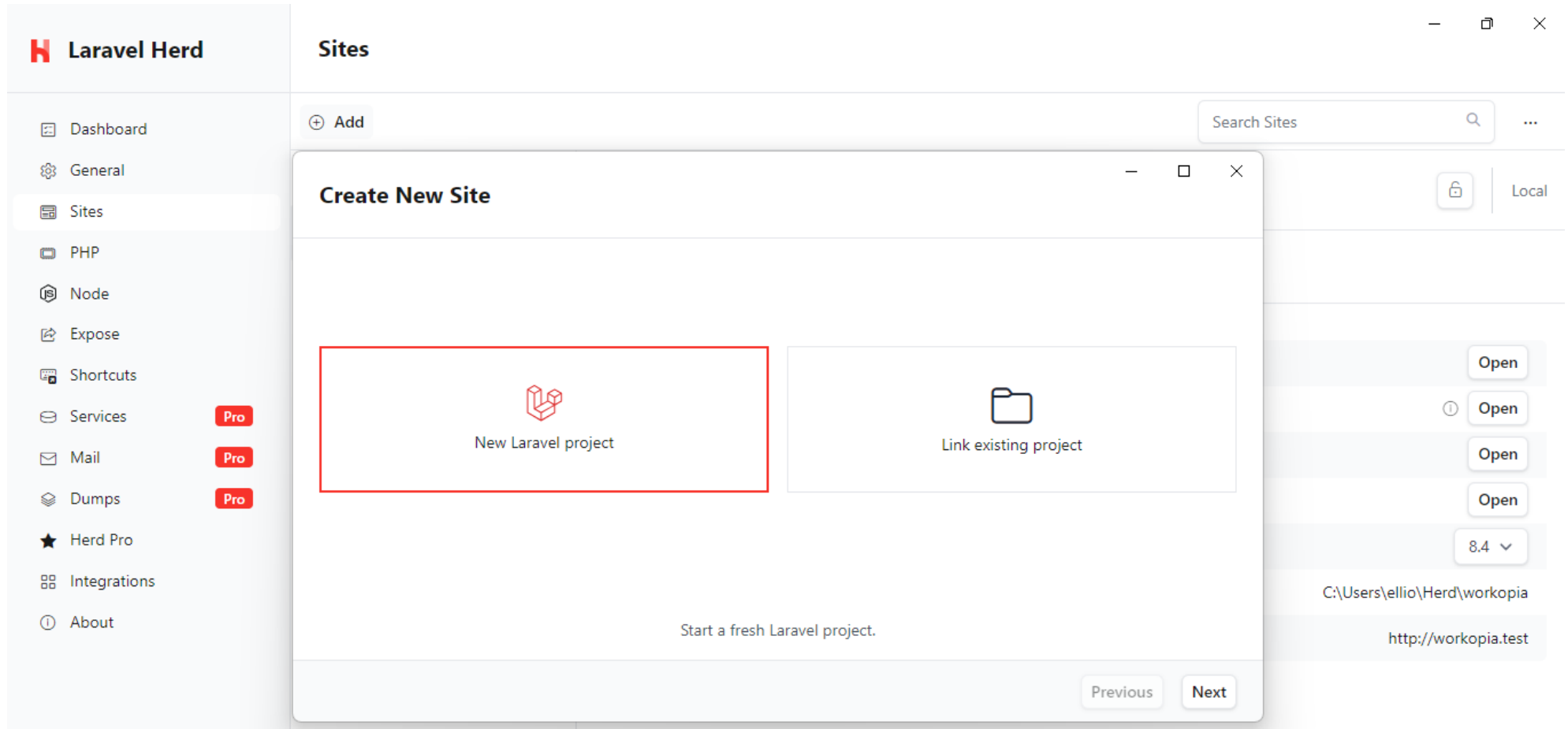
Property	Value
PHP Version	8.4
Path	C:\Users\ellio\Herd\workopia
URL	http://workopia.test

Sites: Where our websites can be found

Php: Chage version of PHP

1.4 Create new site in Herd Dashboard

From the sites tab, click on +add then create new project and then NEXT.



Starter kits are pre built frameworks with add-ons. For this example, we will choose no starter kit then NEXT.

The screenshot shows the Laravel Herd interface. On the left is a sidebar with navigation links: Dashboard, General, Sites (highlighted), PHP, Node, Expose, Shortcuts, Services (Pro), Mail (Pro), Dumps (Pro), Herd Pro, Integrations, and About. The main area is titled 'Sites' and contains an 'Add' button and a search bar. A modal window titled 'Create New Site' is open in the center, displaying five options: 'No starter kit' (highlighted with a red border), 'React', 'Vue', 'Livewire', and 'Custom starter kit'. In the background, a list of sites is visible, including 'Local' and several 'Open' buttons, with a path 'C:\Users\ellio\Herd\workopia' and a URL 'http://workopia.test'.

We can name the site, in this case workapedia. Note the project folder is C:/username/herd. Click on Next and it will generate all the code to produce a local dev environment.

The screenshot shows the 'Laravel Herd' application interface. On the left is a sidebar with navigation links: Dashboard, General, Sites (highlighted), PHP, Node, Expose, Shortcuts, Services (with a 'Pro' badge), Mail (with a 'Pro' badge), Dumps (with a 'Pro' badge), Herd Pro, Integrations, and About. The main area is titled 'Sites' and contains an 'Add' button and a 'Search Sites' input field. A modal window titled 'Create New Site' is open in the center. It has two input fields: 'Project Name' with the text 'workapedia' and 'Target Location' with a dropdown menu showing 'C:\Users\ellio\Herd'. Below these fields is the text 'Your project will be created in this folder.' At the bottom of the modal are 'Previous' and 'Next' buttons. In the background, a list of sites is visible, each with an 'Open' button and a version number (e.g., 8.4).

Laravel Herd

Sites

+ Add

Search Sites

Create New Site

Project Name: workapedia

Target Location: C:\Users\ellio\Herd

Your project will be created in this folder.

Previous Next

Open

Open

Open

Open

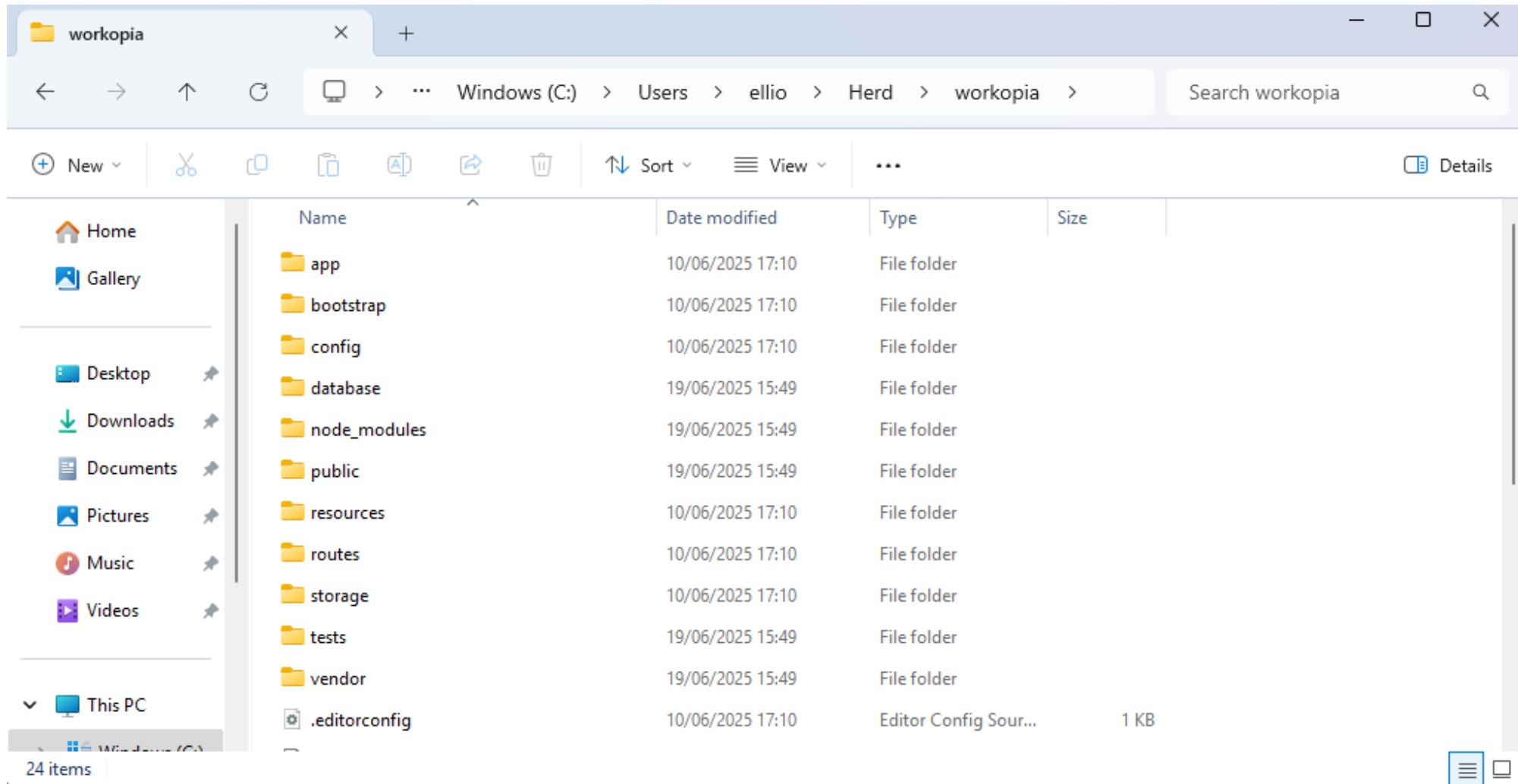
8.4

C:\Users\ellio\Herd\workopia

http://workopia.test

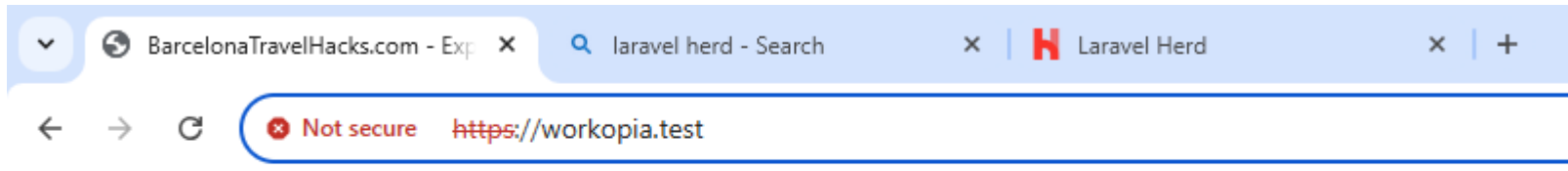
1.5 Open herd site project files in VScode

Note that the folder has been created in the herd directory for our site with all the prebuilt Laravel framework files. Open the root of the site folder, in this case, workopia in Vscode.



1.6 Open project in browser

The site can also be opened in the browser



Observations: Laravel Herd cannot be used with Xampp running on the same base machine due to issues with the hosts.

**Workaround was to install Laravel herd inside a windows 10 virtual machine.
I will decide later if Laravel herd becomes my default dev environment in preference to Xampp.**

2. Getting Started with Laravel Routing

[2.1 Introduction to Routing](#)

[2.2 Disable CSRF for Certain Endpoints](#)

[2.3 Specify allowed methods on an endpoint](#)

[2.4 Named routes](#)

[2.5 API Endpoints](#)

[2.6 List Endpoints with Artisan](#)

[2.7 Passing Parameters in the URL](#)

[2.8 URL Parameter constraints](#)

[2.9 URL Global Parameter constraints](#)

[2.10 Request Object](#)

[2.11 Request Object Query Parameters](#)

[2.12 Response Helper](#)

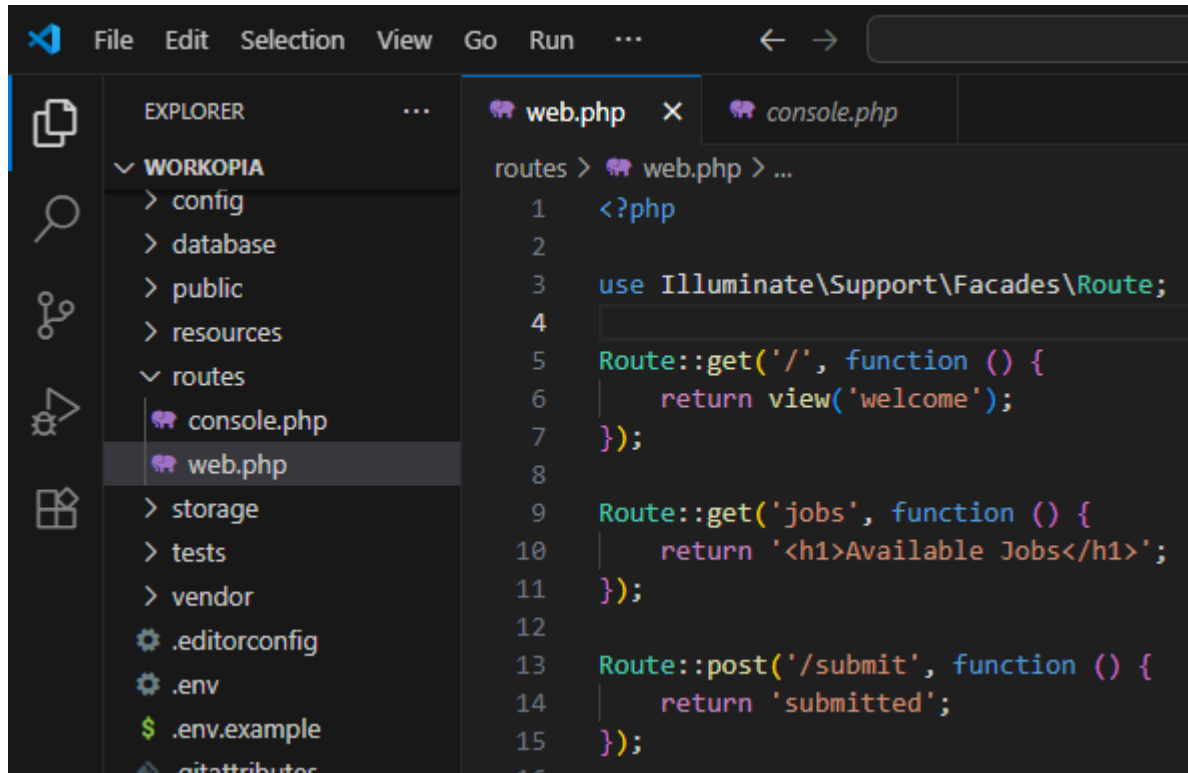
[2.13 Response Header Http Code](#)

[2.14 Response Header Content Type](#)

[2.15 Response Header Download](#)

[2.16 Response Header Cookie](#)

2.1 Introduction to Routing

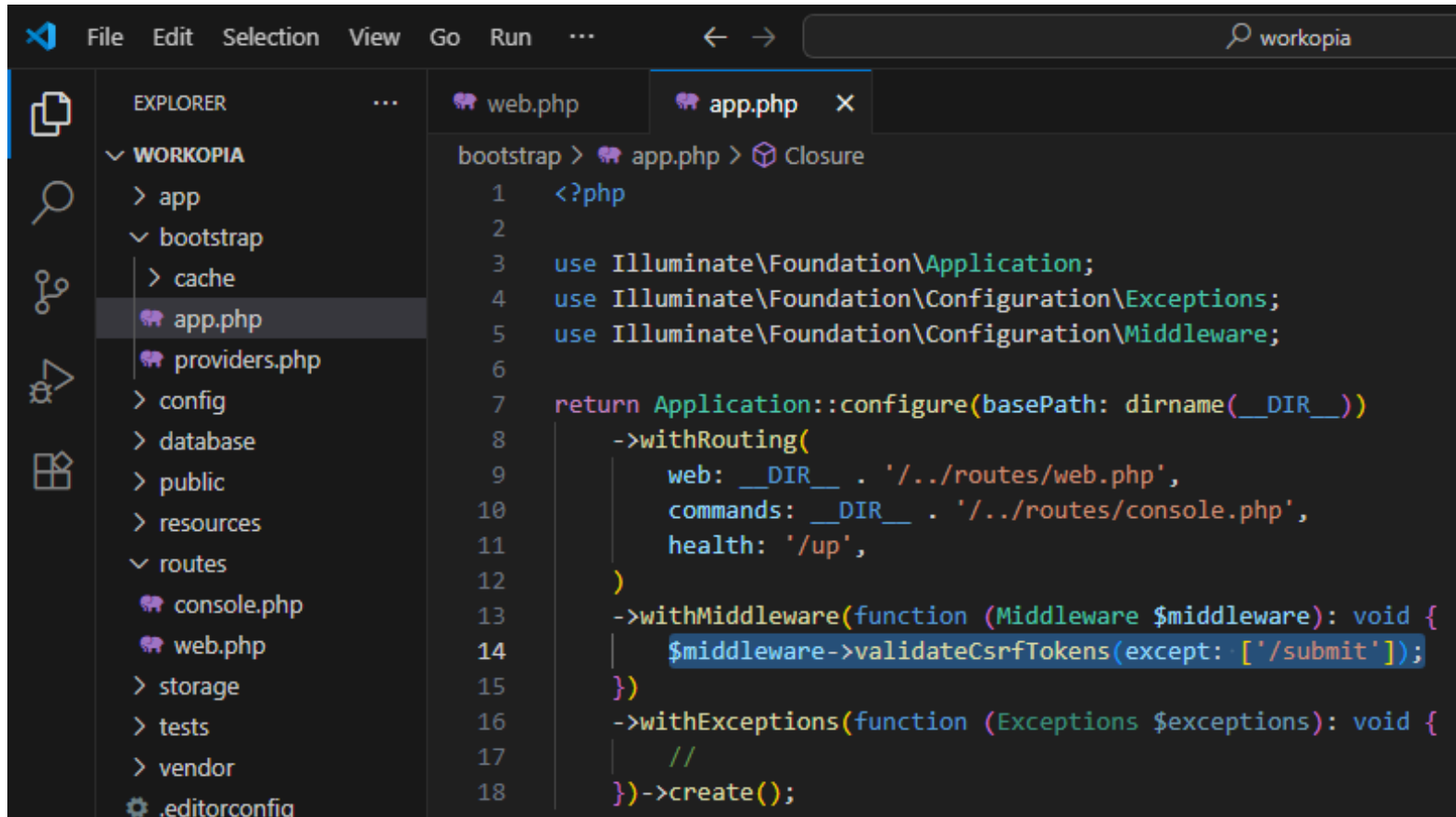


If I add an endpoint of 'jobs' I can return some HTML.

I can also add in a route to handle a POST request to a 'submit' endpoint. Note how the response is a 419. This is because Laravel comes with Cross site request forgery (CSRF) protection already included. CSRF is where a bad actor tries to access a form handler from outside the website.

```
curl -X POST http://workopia.test/submit
<div class="px-4 text-lg text-gray-500 border-r border-gray-400
tracking-wider">419</div>
<div class="ml-4 text-lg text-gray-500 uppercase tracking-wider">
Page Expired</div>
```

2.2 Disable CSRF for Certain Endpoints

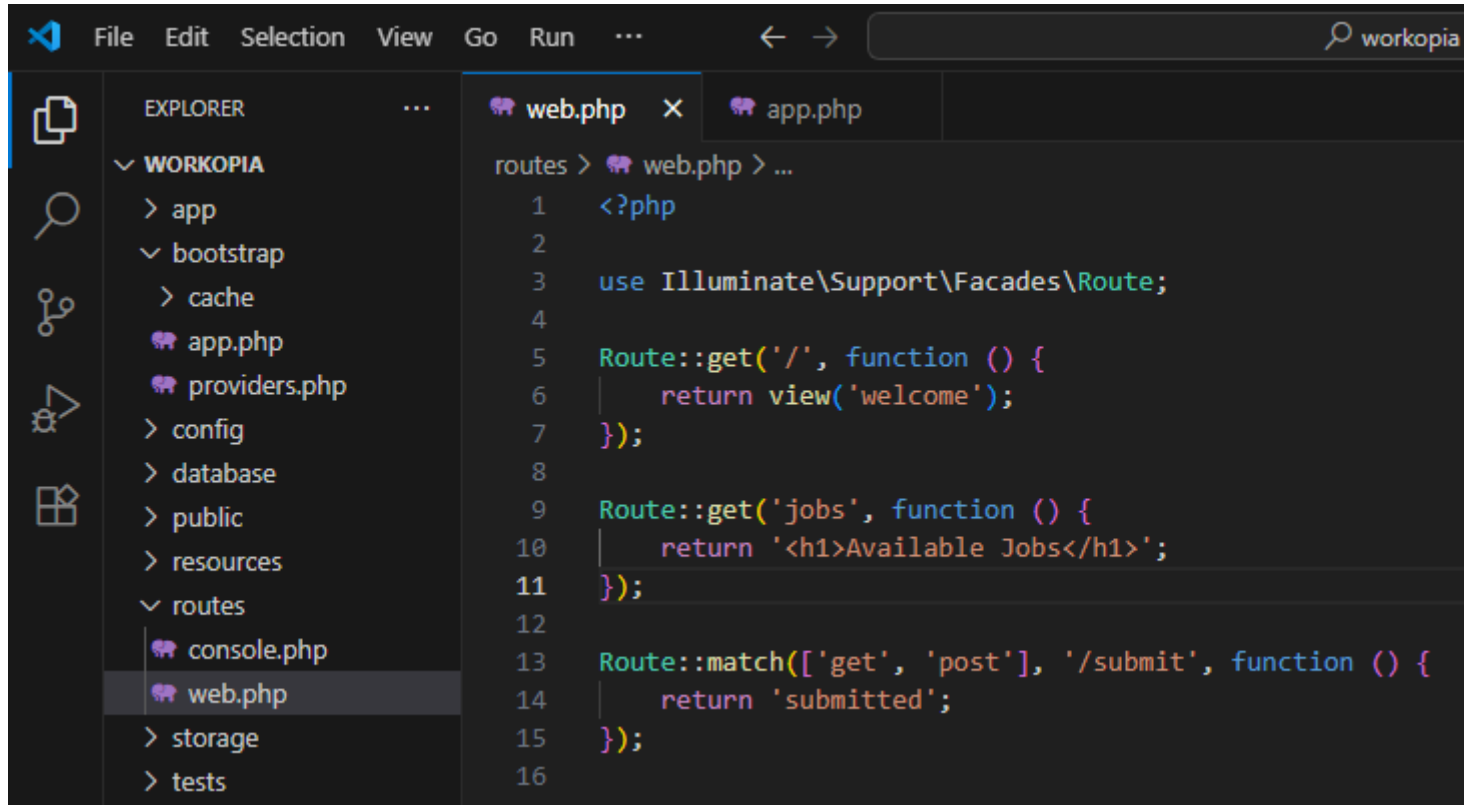


```
bootstrap > app.php > Closure
1  <?php
2
3  use Illuminate\Foundation\Application;
4  use Illuminate\Foundation\Configuration\Exceptions;
5  use Illuminate\Foundation\Configuration\Middleware;
6
7  return Application::configure(basePath: dirname(__DIR__))
8      ->withRouting(
9          web: __DIR__ . '/../routes/web.php',
10         commands: __DIR__ . '/../routes/console.php',
11         health: '/up',
12     )
13     ->withMiddleware(function (Middleware $middleware): void {
14         $middleware->validateCsrfTokens(except: ['/submit']);
15     })
16     ->withExceptions(function (Exceptions $exceptions): void {
17         //
18     }->create();
```

In the bootstrap/app.php file I can add a middleware validate CSRF exception on the 'submit' endpoint. Now when I do a curl request to the endpoint it returns submitted as defined in the web.php file.

```
curl -X POST http://workopia.test/submit
Submitted
```

2.3 Specify Allowed Methods on an Endpoint



```
File Edit Selection View Go Run ... workopia

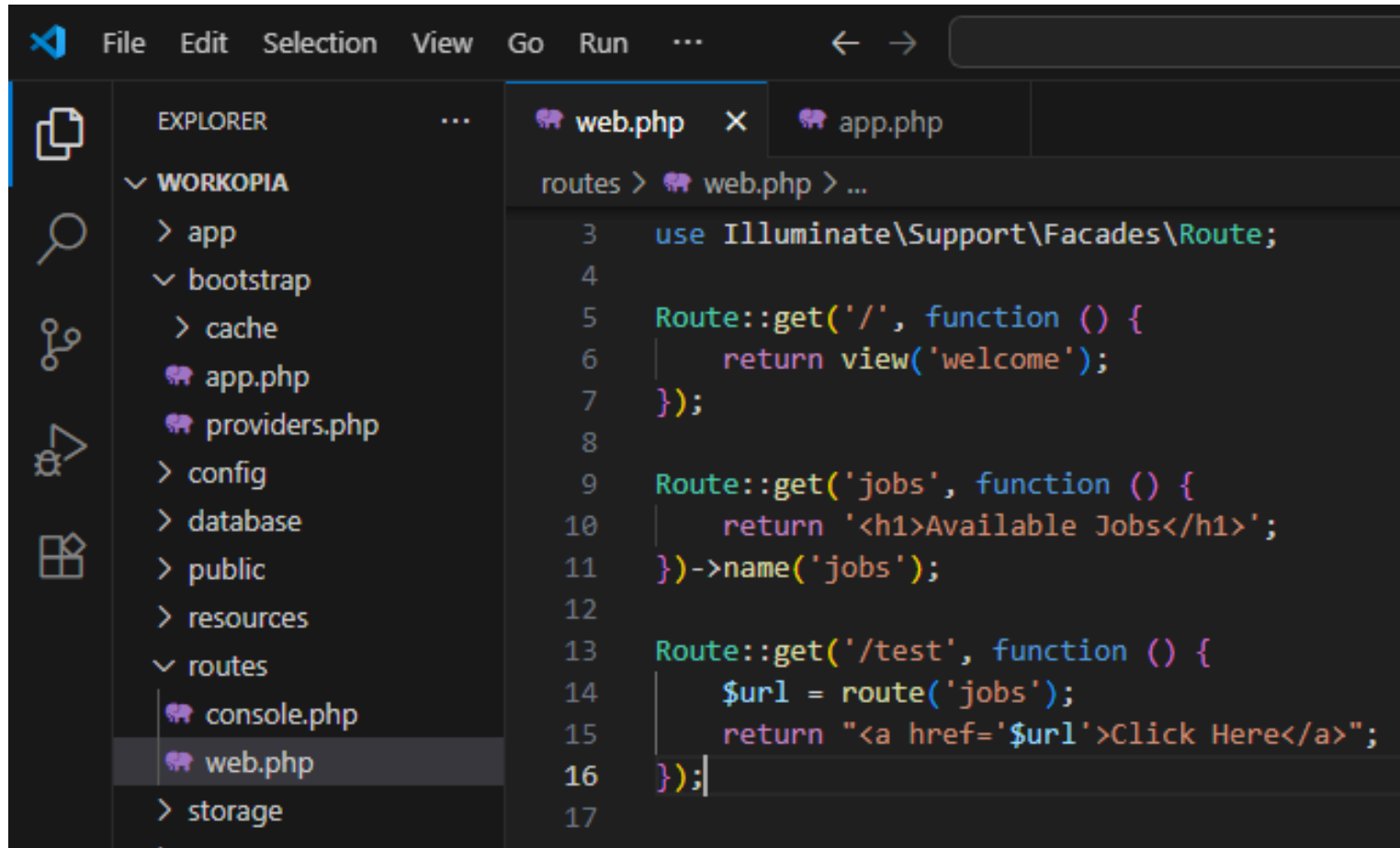
EXPLORER
WORKOPIA
  > app
  > bootstrap
  > cache
  app.php
  providers.php
  > config
  > database
  > public
  > resources
  routes
    console.php
    web.php
  > storage
  > tests

routes > web.php > ...
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/', function () {
6     return view('welcome');
7 });
8
9 Route::get('jobs', function () {
10     return '<h1>Available Jobs</h1>';
11 });
12
13 Route::match(['get', 'post'], '/submit', function () {
14     return 'submitted';
15 });
16
```

By adding optional parameters in the ROUTE function I can specify the allowed methods for an endpoint. In this example the 'submit' endpoint is only allowing GET or POST. A PUT would not work.

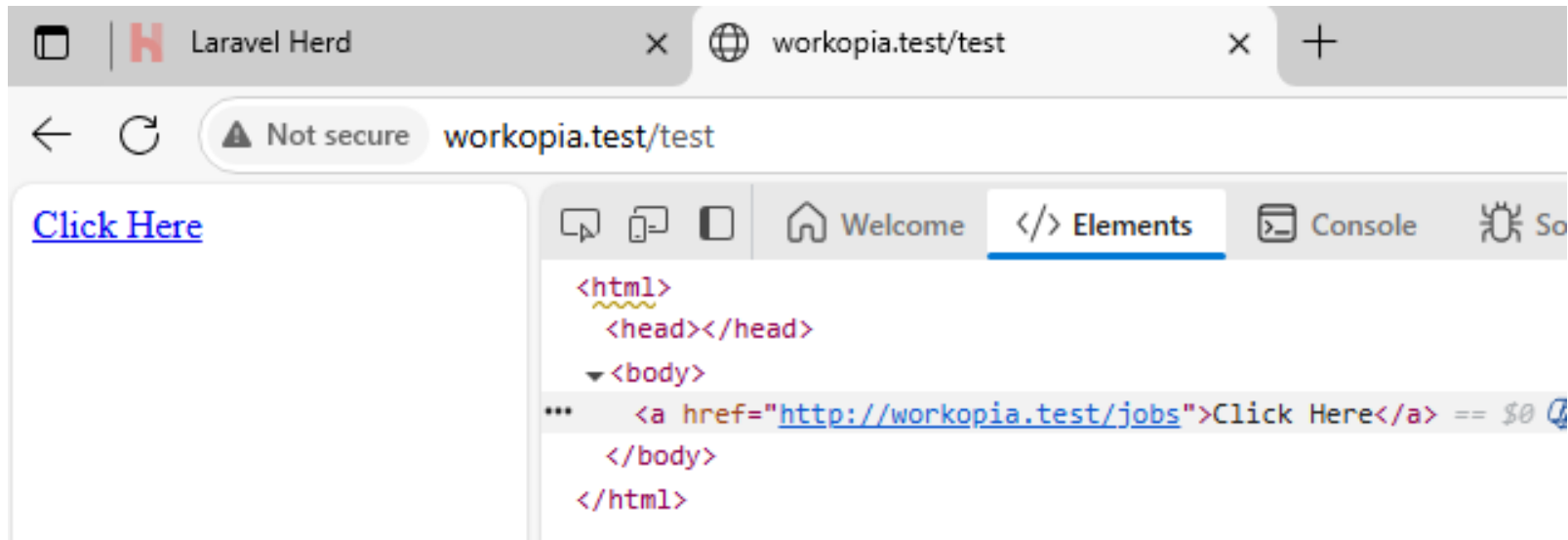
Changing match to 'any' and removing ['get','post'] would allow any kind of request method.

2.4 Named Routes

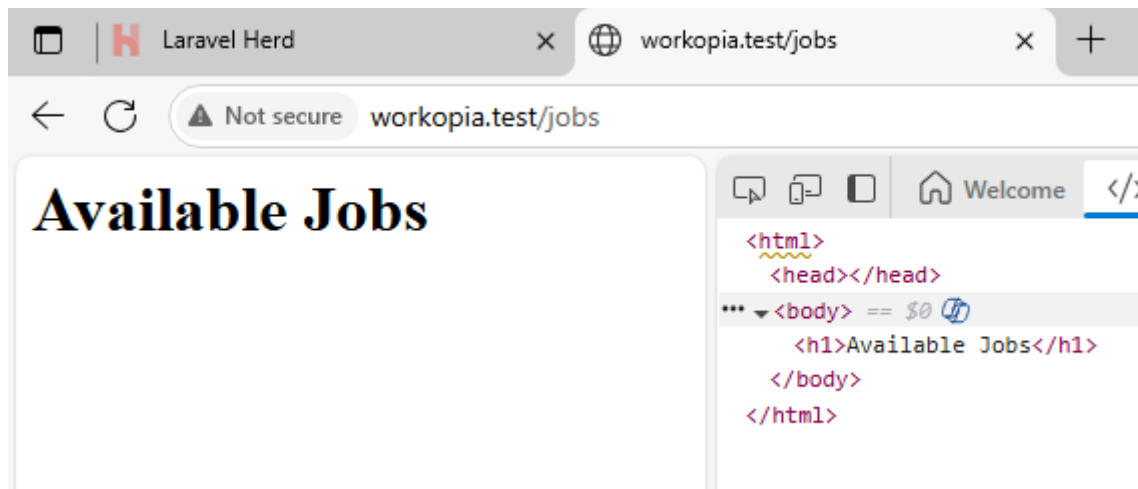
A screenshot of a code editor interface. On the left is a file explorer with a tree view under 'WORKOPIA'. It contains folders 'app', 'bootstrap', 'cache', 'config', 'database', 'public', 'resources', and 'storage'. There are also files 'app.php', 'providers.php', 'console.php', and 'web.php'. 'web.php' is selected. The main editor area shows the code for 'web.php'. The code uses the Illuminate\Support\Facades\Route facade to define three routes: a root route, a 'jobs' route, and a '/test' route that uses the 'jobs' route's name. The code is as follows:

```
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/', function () {
6     return view('welcome');
7 });
8
9 Route::get('jobs', function () {
10     return '<h1>Available Jobs</h1>';
11 })->name('jobs');
12
13 Route::get('/test', function () {
14     $url = route('jobs');
15     return "<a href='$url'>Click Here</a>";
16 });
17
```

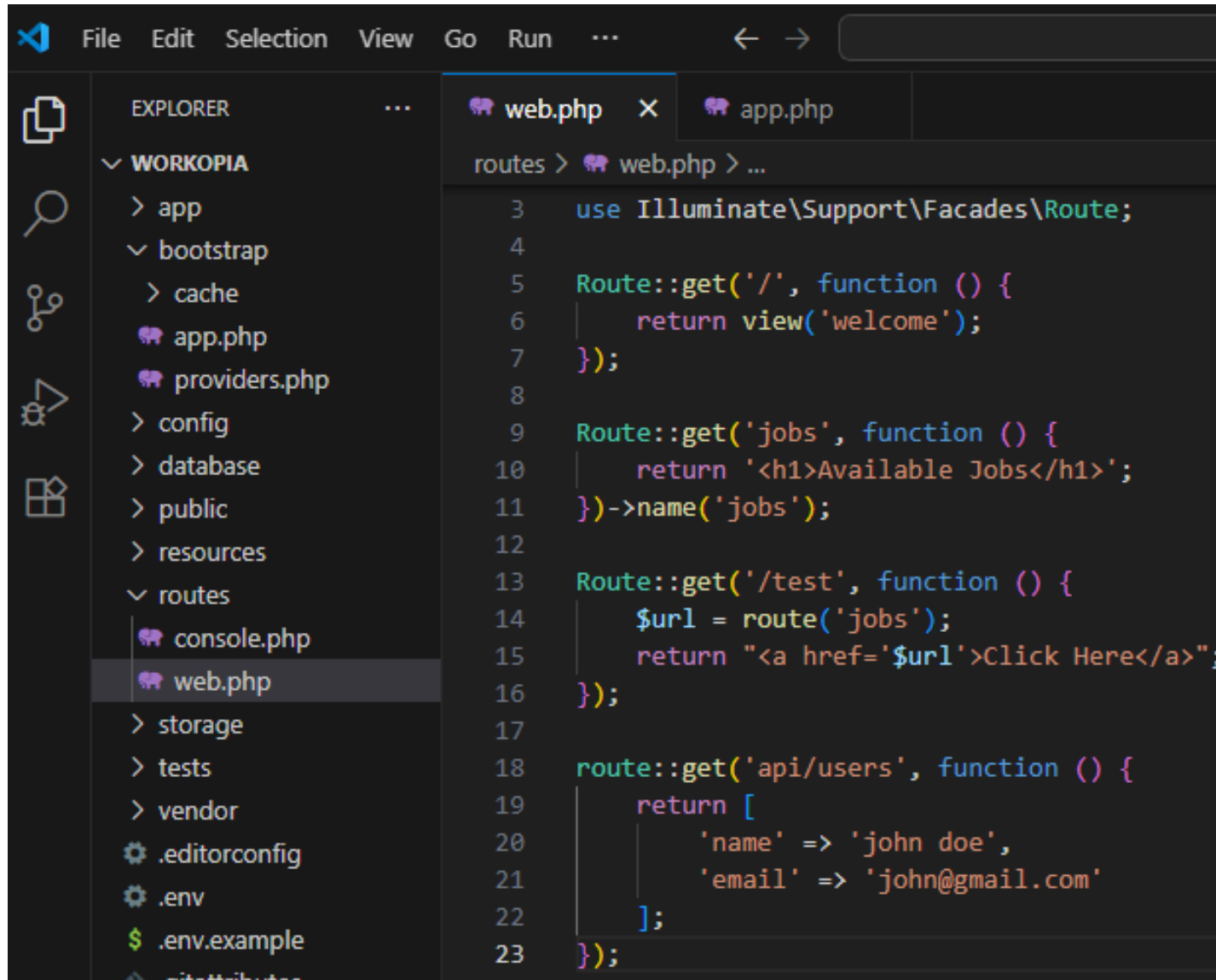
I can add name an endpoint which allows me to use it somewhere else. For example, I can have an endpoint of '/test' that will route to '/jobs'.



Note how /test now contains html markup for a hyperlink pointing to the '/jobs' endpoint which does route correctly to that endpoint.



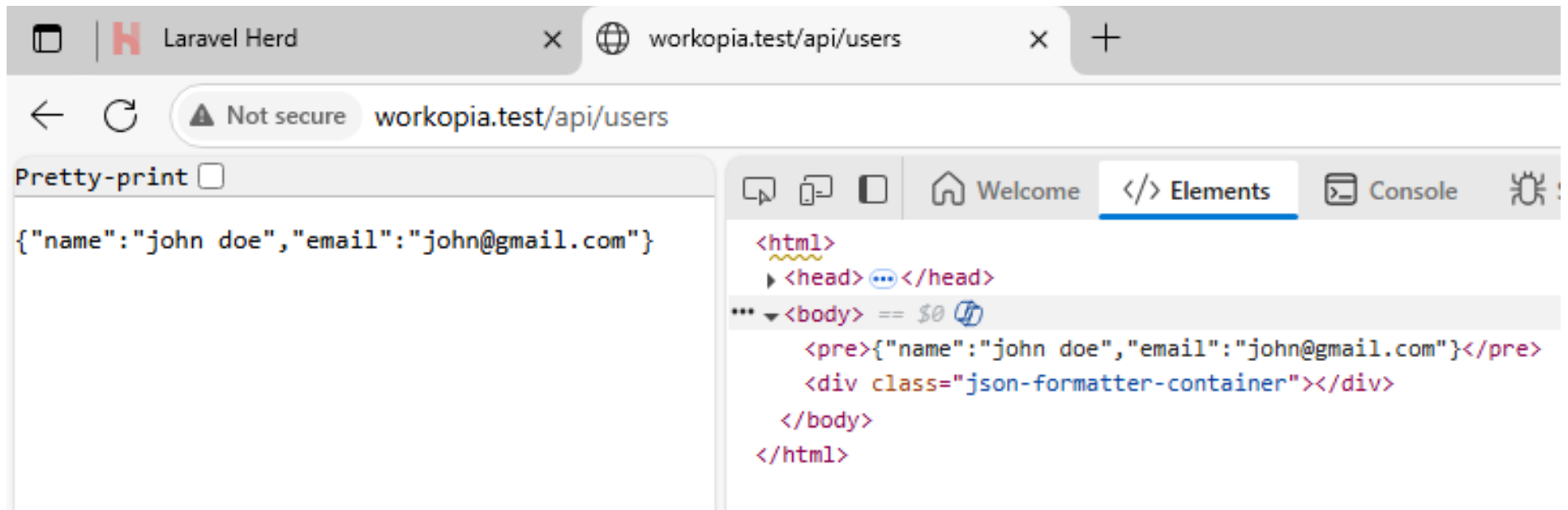
2.5 API Endpoints



The screenshot shows a code editor with a dark theme. On the left is the 'EXPLORER' sidebar showing a file tree for a project named 'WORKOPIA'. The tree includes folders like 'app', 'bootstrap', 'cache', 'config', 'database', 'public', 'resources', 'routes', 'storage', 'tests', and 'vendor'. The 'routes' folder is expanded, showing 'console.php' and 'web.php'. The 'web.php' file is selected. The main editor area shows the contents of 'web.php', which defines three routes using the Laravel Illuminate routing facade. The first route is for the root path ('/'), returning a 'welcome' view. The second route is for '/jobs', returning an HTML snippet. The third route is for '/test', which dynamically generates a link to the '/jobs' endpoint. Additionally, there is a route for 'api/users' that returns an associative array with user details.

```
File Edit Selection View Go Run ...  
EXPLORER  
WORKOPIA  
  > app  
  > bootstrap  
    > cache  
    app.php  
    providers.php  
  > config  
  > database  
  > public  
  > resources  
  > routes  
    console.php  
    web.php  
  > storage  
  > tests  
  > vendor  
  .editorconfig  
  .env  
  .env.example  
  gitattributes  
web.php  
routes > web.php > ...  
3 use Illuminate\Support\Facades\Route;  
4  
5 Route::get('/', function () {  
6     return view('welcome');  
7 });  
8  
9 Route::get('jobs', function () {  
10     return '<h1>Available Jobs</h1>';  
11 })->name('jobs');  
12  
13 Route::get('/test', function () {  
14     $url = route('jobs');  
15     return "<a href='$url'>Click Here</a>";  
16 });  
17  
18 route::get('api/users', function () {  
19     return [  
20         'name' => 'john doe',  
21         'email' => 'john@gmail.com'  
22     ];  
23 });
```

In this simple example the endpoint is returning an associative array.



2.6 list Endpoints with Artisan

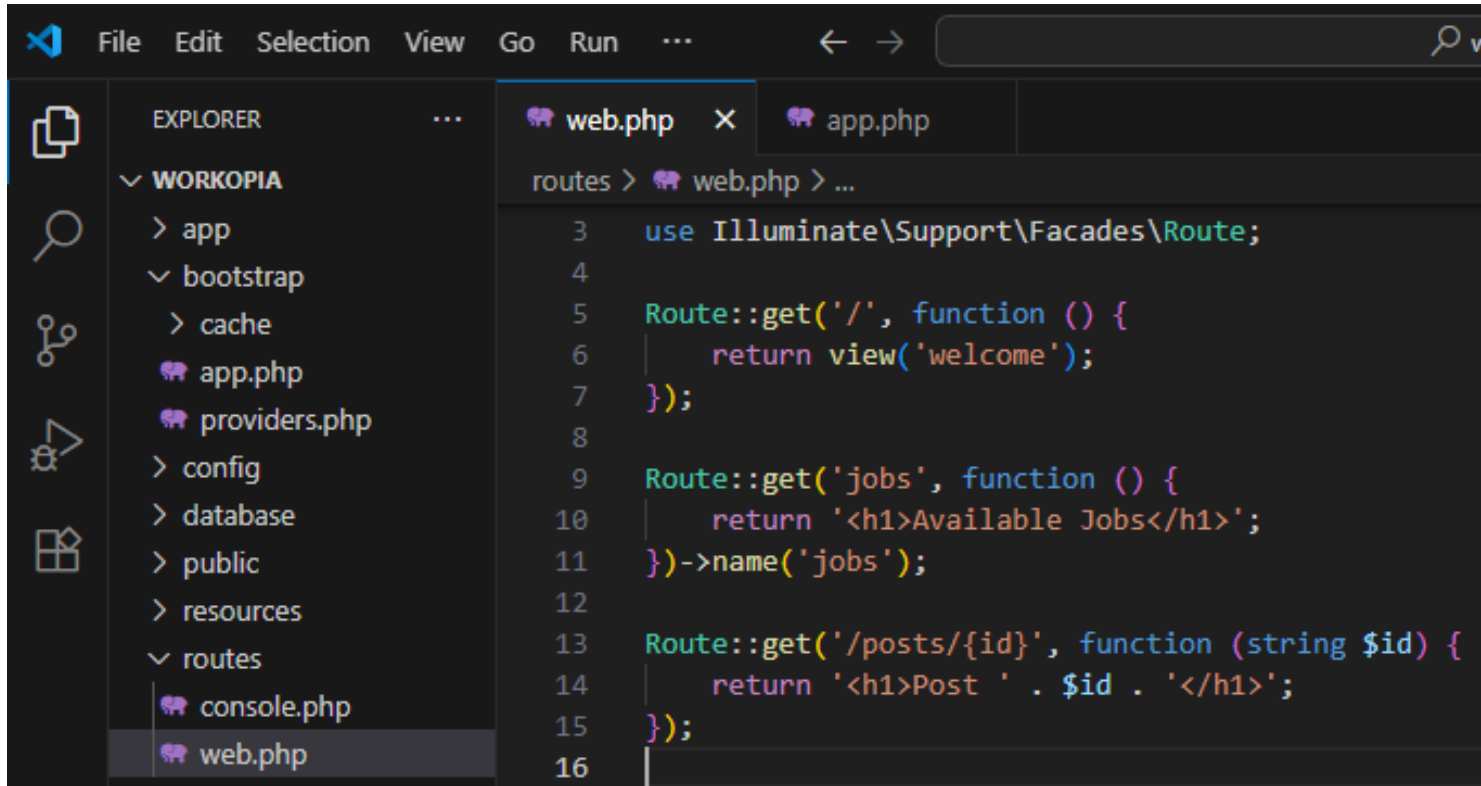
From the root of the project folder the 'php artisan route:list' command can be used to list all endpoints.

```
C:\Users\ellio\Herd\workopia>php artisan route:list
```

GET HEAD	/.....
GET HEAD	api/users.....
GET HEAD	jobs jobs
GET HEAD	storage/{path} storage.local
GET HEAD	test
GET HEAD	up

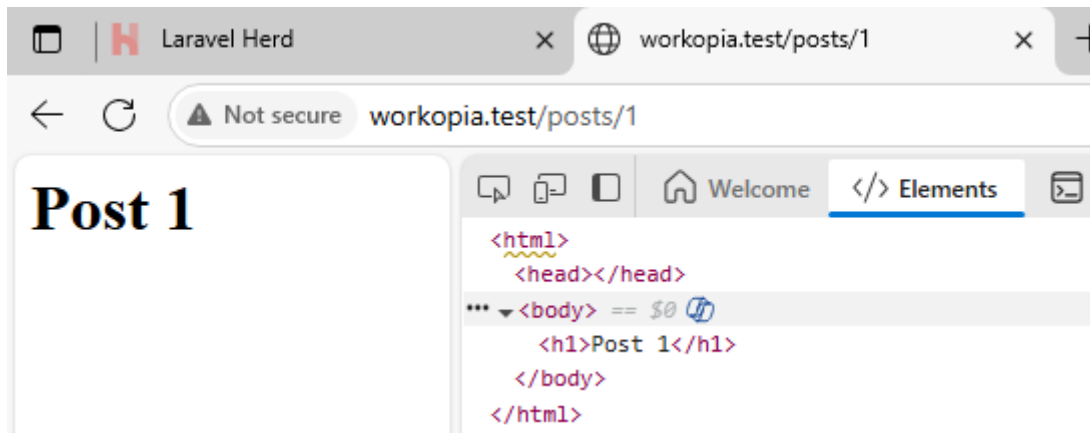
Showing [6] routes

2.7 Passing Parameters in the URL

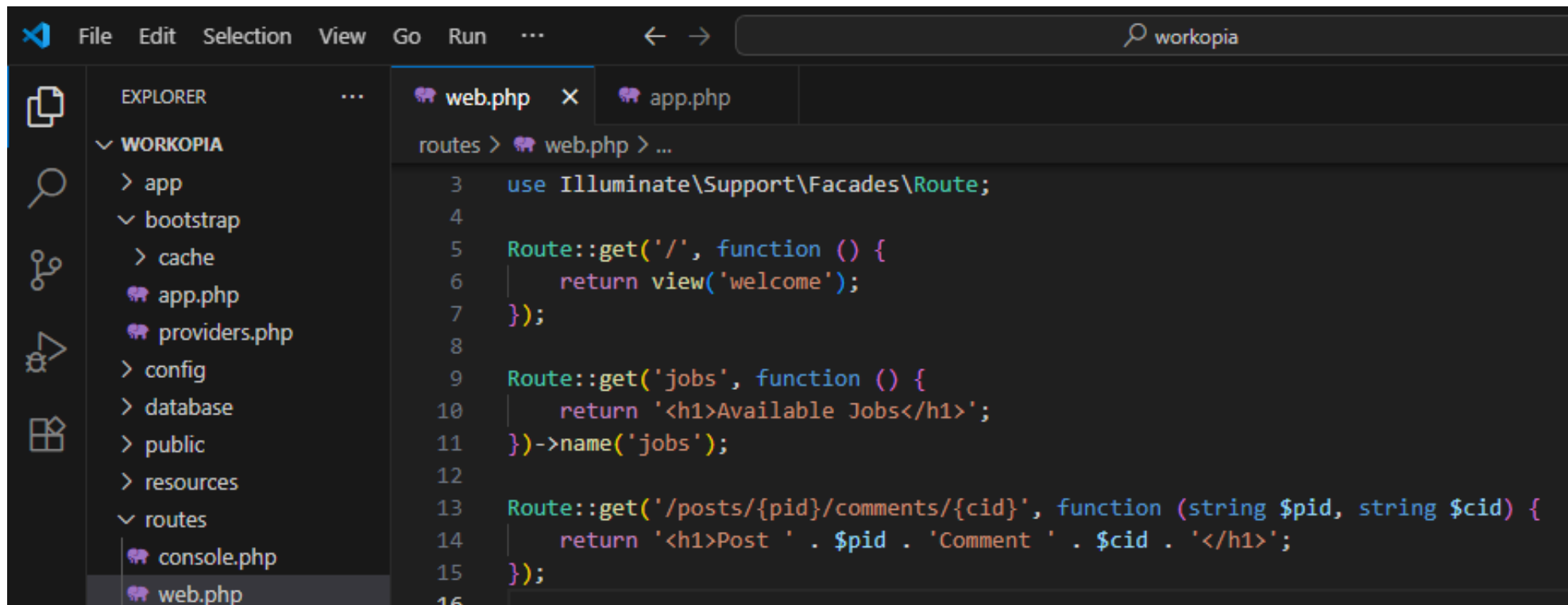


The screenshot shows the Visual Studio Code editor with the Laravel project structure on the left. The 'routes' directory is expanded, showing 'console.php' and 'web.php'. The 'web.php' file is open in the editor, showing the following code:

```
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/', function () {
6     return view('welcome');
7 });
8
9 Route::get('jobs', function () {
10     return '<h1>Available Jobs</h1>';
11 })->name('jobs');
12
13 Route::get('/posts/{id}', function (string $id) {
14     return '<h1>Post ' . $id . '</h1>';
15 });
16
```

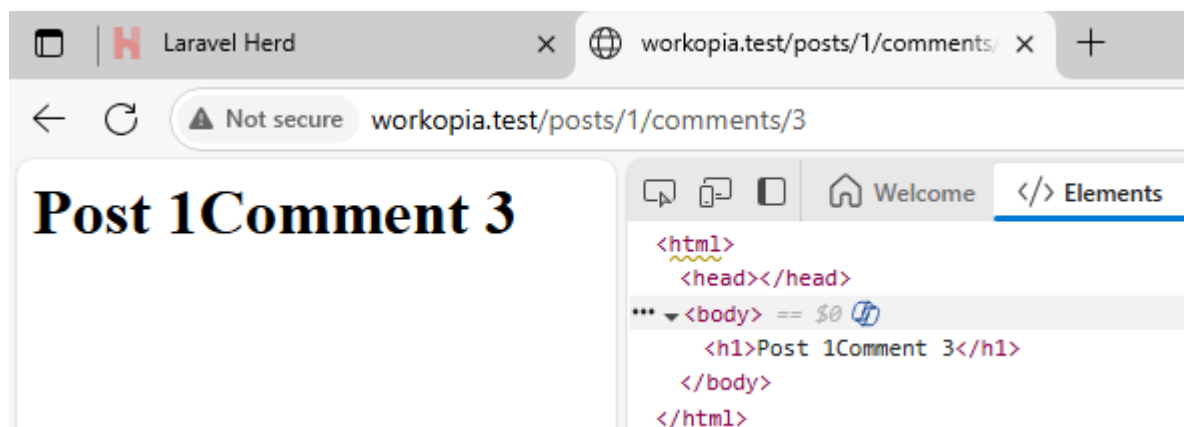


I now have an endpoint of 'posts' that takes an additional parameter of an id value which is a string.



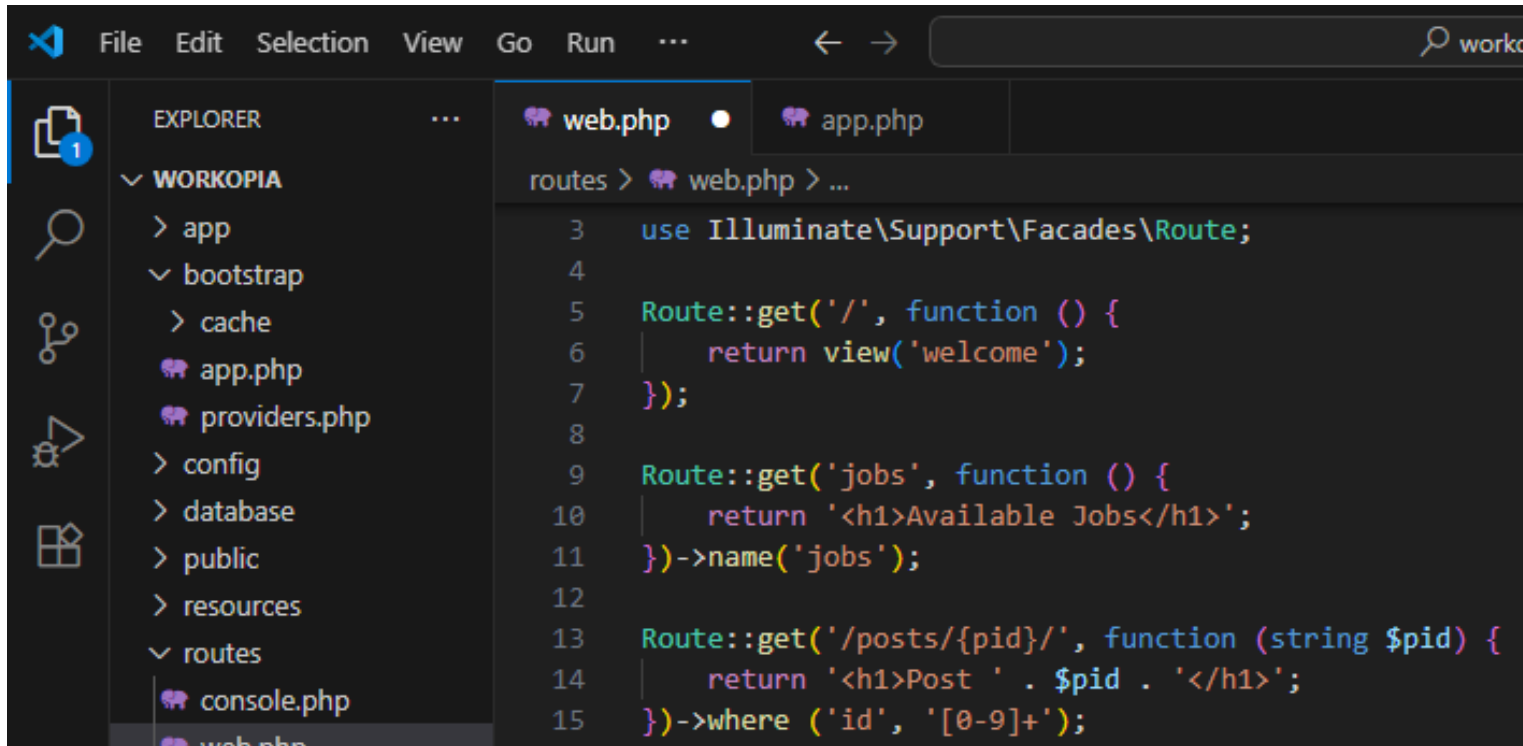
The image shows a Visual Studio Code editor window. On the left is the Explorer sidebar showing a project named 'WORKOPIA' with a 'routes' directory containing 'console.php' and 'web.php'. The 'web.php' file is open in the editor. The code in 'web.php' defines three routes using the Laravel Route facade:

```
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/', function () {
6     return view('welcome');
7 });
8
9 Route::get('jobs', function () {
10     return '<h1>Available Jobs</h1>';
11 }->name('jobs');
12
13 Route::get('/posts/{pid}/comments/{cid}', function (string $pid, string $cid) {
14     return '<h1>Post ' . $pid . ' Comment ' . $cid . '</h1>';
15 });
16
```



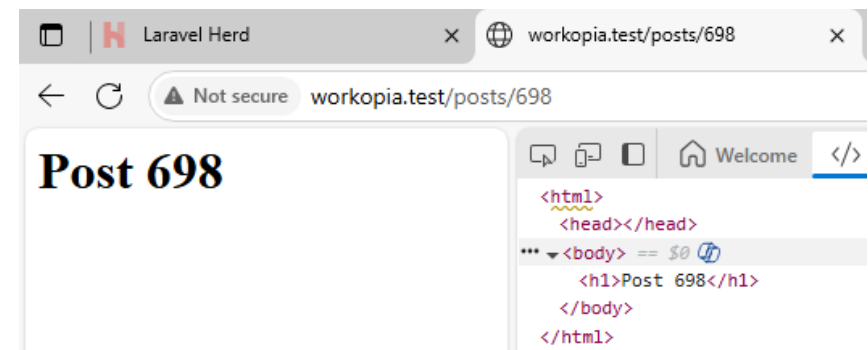
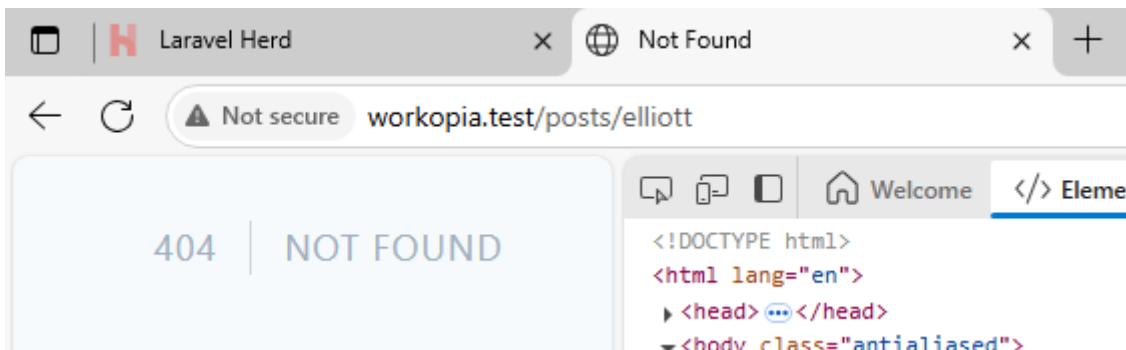
I can also pass multiple parameters in the URL.

2.8 URL Parameter Constraints



```
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/', function () {
6     return view('welcome');
7 });
8
9 Route::get('jobs', function () {
10     return '<h1>Available Jobs</h1>';
11 }->name('jobs');
12
13 Route::get('/posts/{pid}/', function (string $pid) {
14     return '<h1>Post ' . $pid . '</h1>';
15 }->where('id', '[0-9]+');
```

I can use where to specify that the post 'pid' must be a number of any length of characters from 0 to 9. i.e. 1,2...9, 11,...19, 400 671 etc. Alphanumeric characters in the 'pid' parameter will not work.



I also use the regex to define only characters in the URL parameter. [a-zA-z]+

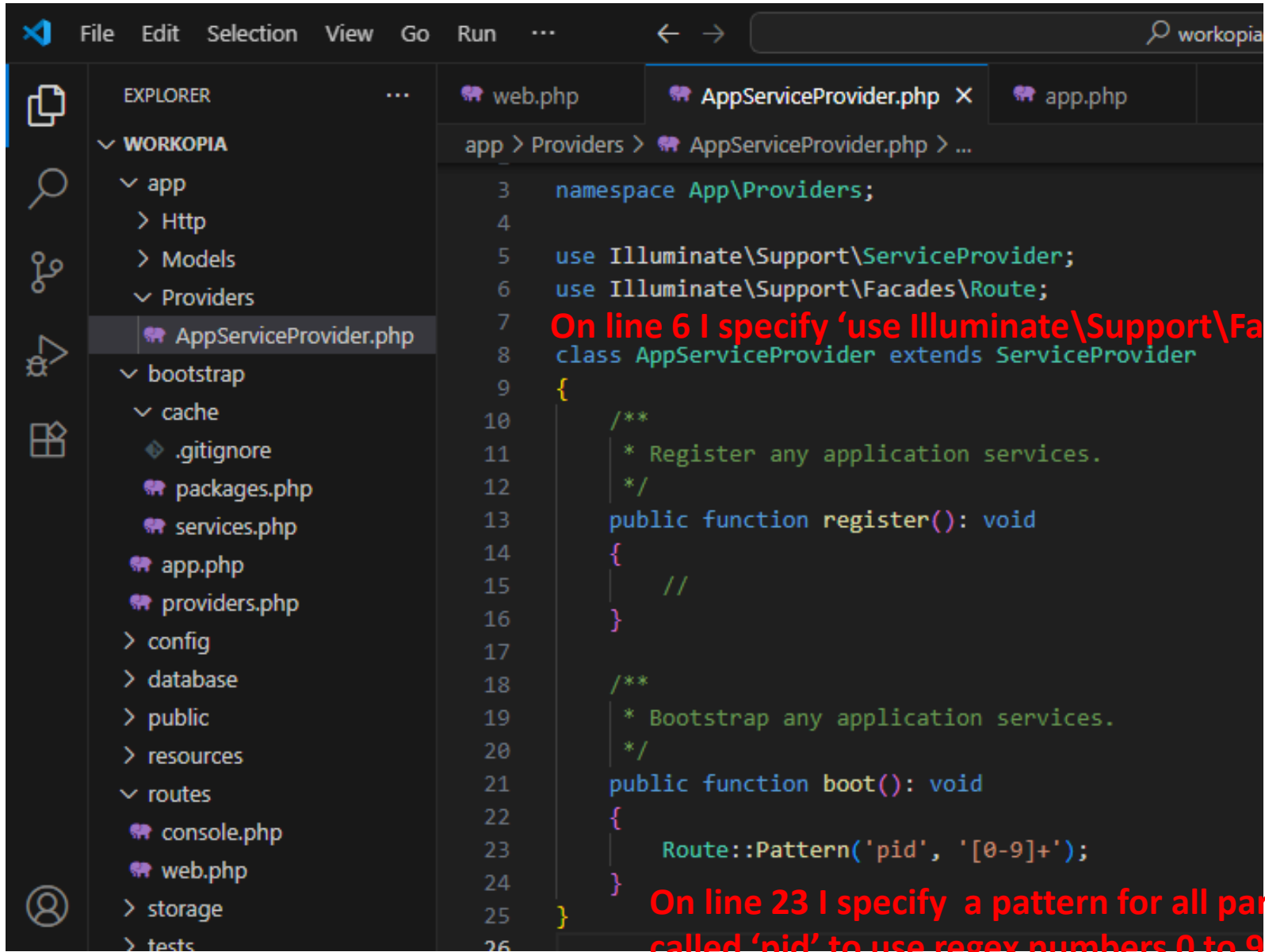
```
13 Route::get('/posts/{pid}/', function (string $pid) {
14 |     return '<h1>Post ' . $pid . '</h1>';
15 | });->whereNumber('pid');
```

To avoid using regex, I can use the 'whereNumber' and only specify the parameter.

```
Route::get('/posts/{pid}/', function (string $pid) {
|     return '<h1>Post ' . $pid . '</h1>';
| });->whereAlpha('pid');
```

I can use 'whereAlpha' to restrict the parameter to letters only.

2.9 URL Parameter Global Constraints



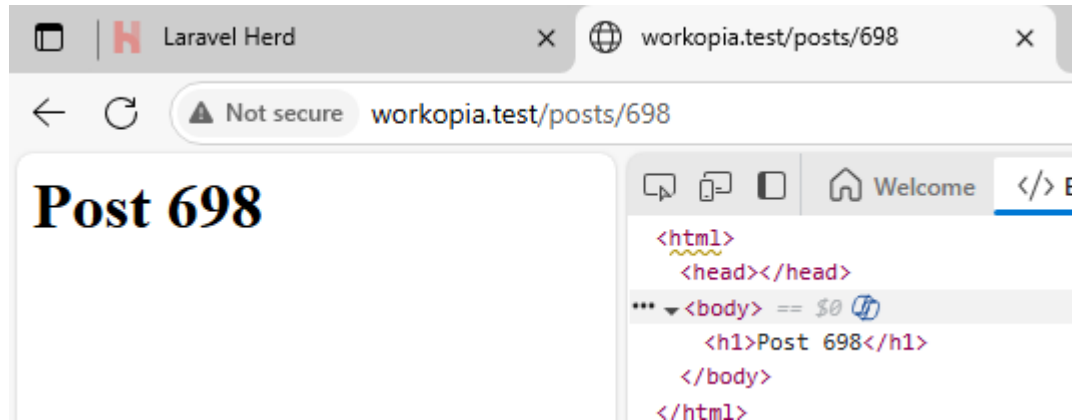
```
File Edit Selection View Go Run ...  
workopia  
EXPLORER  
WORKOPIA  
  app  
    Http  
    Models  
    Providers  
      AppServiceProvider.php  
  bootstrap  
  cache  
    .gitignore  
    packages.php  
    services.php  
    app.php  
    providers.php  
  config  
  database  
  public  
  resources  
  routes  
    console.php  
    web.php  
  storage  
  tests  
web.php AppServiceProvider.php X app.php  
app > Providers > AppServiceProvider.php > ...  
3 namespace App\Providers;  
4  
5 use Illuminate\Support\ServiceProvider;  
6 use Illuminate\Support\Facades\Route;  
7  
8 class AppServiceProvider extends ServiceProvider  
9 {  
10     /**  
11      * Register any application services.  
12      */  
13     public function register(): void  
14     {  
15         //  
16     }  
17  
18     /**  
19      * Bootstrap any application services.  
20      */  
21     public function boot(): void  
22     {  
23         Route::Pattern('pid', '[0-9]+');  
24     }  
25 }  
26
```

On line 6 I specify 'use Illuminate\Support\Facades\Route'

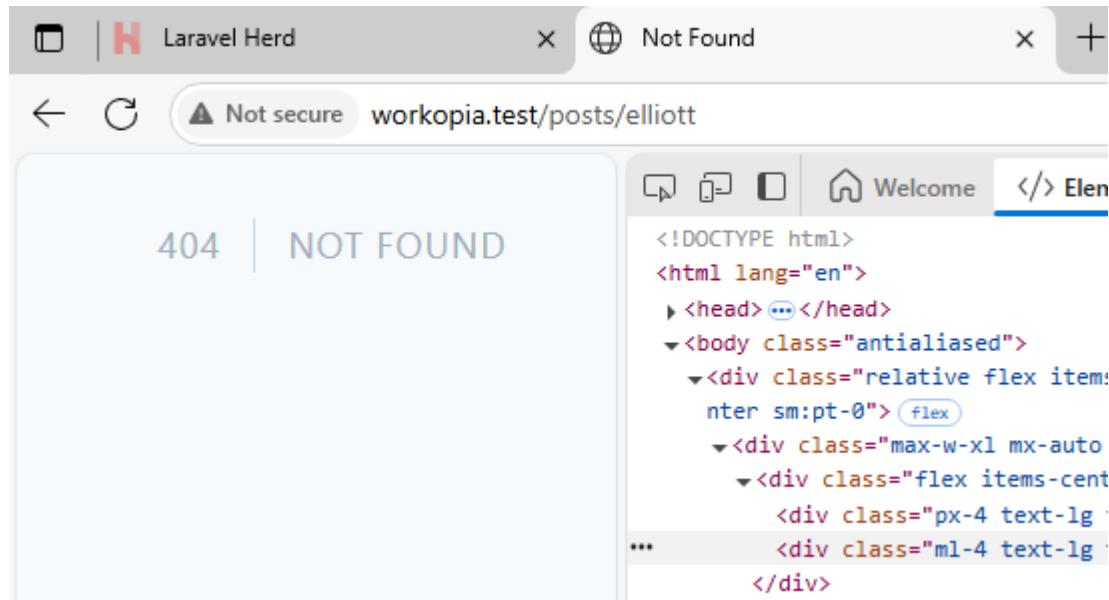
On line 23 I specify a pattern for all parameters called 'pid' to use regex numbers 0 to 9.

```
Route::get('/posts/{pid}/', function (string $pid) {  
    return '<h1>Post ' . $pid . '</h1>';  
});
```

Now I do not need the where clause on the 'pid' because it is defined as a global pattern match



And where 'pid' is a number, it routes correctly.



But where it is alpha it does not route and lands on a 404.

2.10 Request Object

The request object can be very useful and I can get a lot of parameters from it.

1. Request method
2. Request URI
3. Request Headers
4. Request body
5. Query Parameters
6. Form data
7. Uploaded files
8. Cookies
9. Session Data



EXPLORER

WORKOPIA

> config

> database

> public

> resources

> routes

console.php

web.php

> storage

> tests

> vendor

.editorconfig

.env

.env.example

.gitattributes

.gitignore

artisan

composer.json

composer.lock

package.json

phpunit.xml

README.md

web.php

AppServiceProvider.php

app.php

routes > web.php > Closure

1 <?php

2

3 use Illuminate\Support\Facades\Route;

4 use Illuminate\Http\Request;

5 I add (Illuminate) the inbuilt http request header facade

6 Route::get('/', function () {

7 | return view('welcome');

8 }); Laravel will inject a dependencies class to

9 handle all the http header parameters.

10 Route::get('jobs', function () {

11 | return '<h1>Available Jobs</h1>';

12 }->name('jobs');

13

14 Route::get('/test', function (Request \$request) {

15 | return [

16 | 'method' => \$request->method(),

17 | 'url' => \$request->url(),

18 | 'path' => \$request->path(),

19 | 'fullUrl' => \$request->fullUrl(),

20 | 'ip' => \$request->ip(),

21 | 'userAgent' => \$request->userAgent(),

22 | 'header' => \$request->header(),

23 |];

24 });

Returning
an array of
some of the
http
request
header
values.

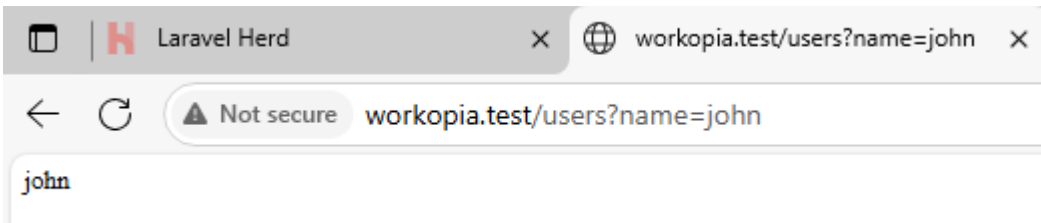
The output is a JSON object of http request header values.

```
{
  "method": "GET",
  "url": "http://workopia.test/test",
  "path": "test",
  "fullUrl": "http://workopia.test/test",
  "ip": "127.0.0.1",
  "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/137.0.0.0 Safari/537.36 Edg/137.0.0.0",
  "header": {
    "cookie": [
      "XSRF-"
    ],
    "accept-language": [
      "en-GB,en;q=0.9,en-US;q=0.8"
    ],
    "accept-encoding": [
      "gzip, deflate"
    ],
    "accept": [
      "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7"
    ],
    "user-agent": [
      "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/137.0.0.0 Safari/537.36 Edg/137.0.0.0"
    ],
    "upgrade-insecure-requests": [
      "1"
    ],
    "connection": [
      "keep-alive"
    ],
    "host": [
      "workopia.test"
    ]
  }
}
```

2.11 Request Object Query Parameters

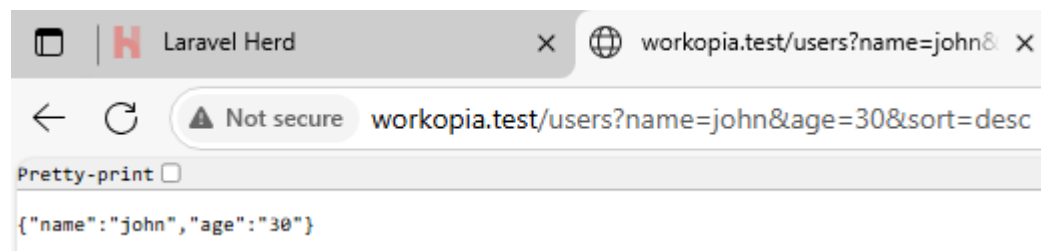
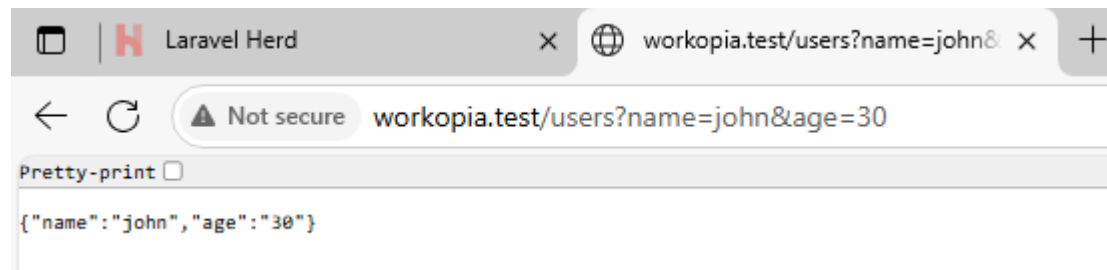
```
Route::get('/users', function (Request $request) {  
    return $request->query('name');  
});
```

If I want to get a query parameter for name from a uri 'users?name=john', I can use the query keyword.



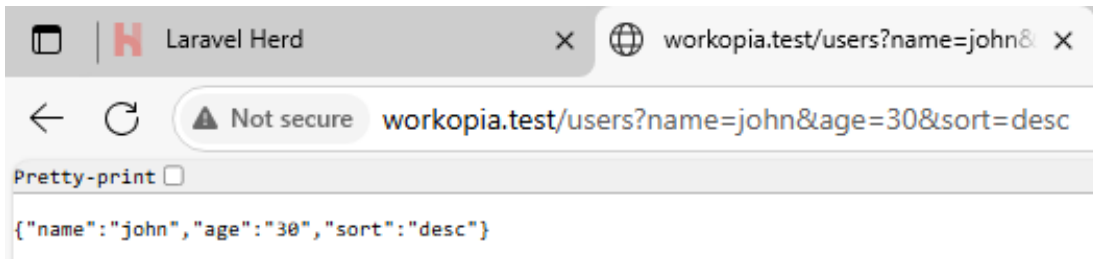
```
Route::get('/users', function (Request $request) {  
    return $request->only('name', 'age');  
});
```

If I want to pass in multiple parameters, I can use the only keyword i.e. from a uri 'users?name=john&age=30'



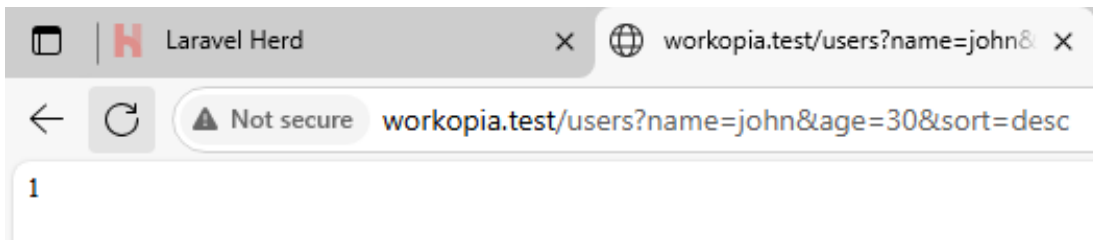
If I pass in more parameters than defined then it does not give me that. i.e. from a uri 'users?name=john&age=30&sort=desc'

```
Route::get('/users', function (Request $request) {  
    return $request->all();  
});
```



If I want all the query parameters without specifying them manually then I can use the 'all' keyword. i.e. from a uri 'users?name=john&age=30&sort=desc'

```
Route::get('/users', function (Request $request) {  
    return $request->has('name');  
});
```



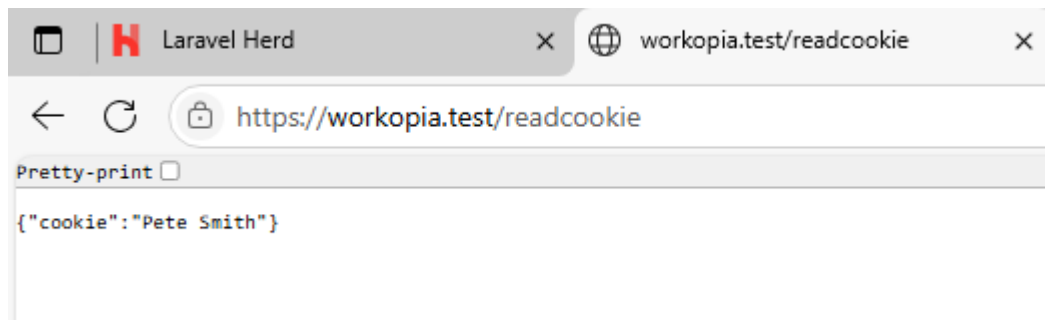
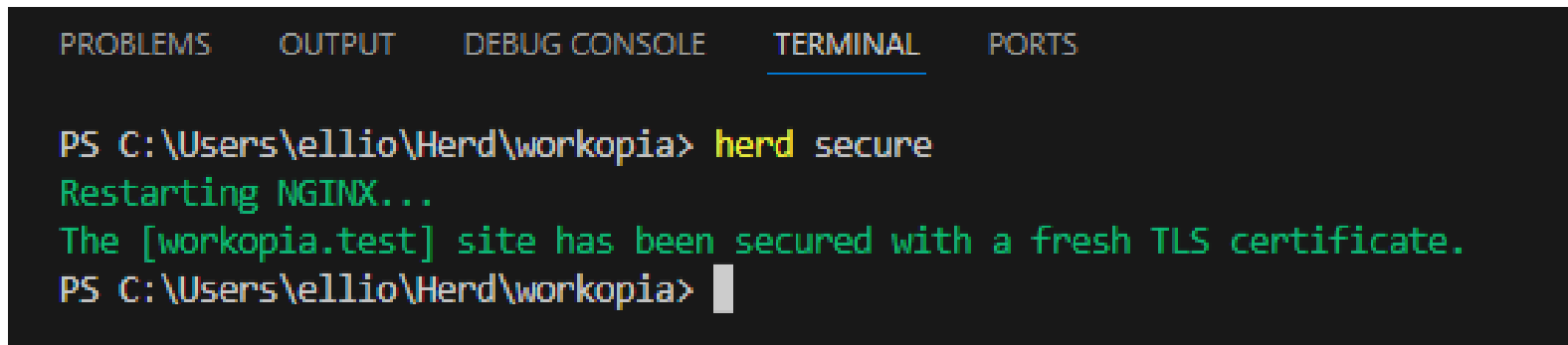
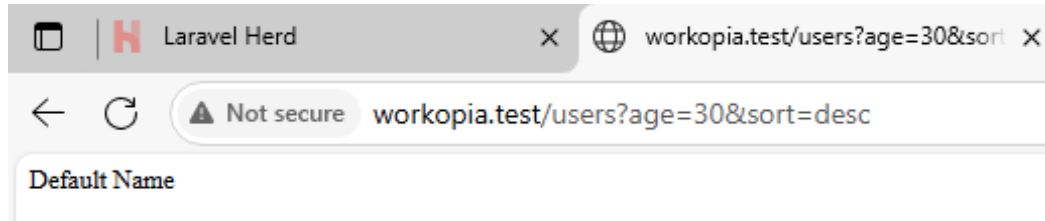
The has keyword will return a Boolean (1) for true or nothing if false when the named query parameter is in the uri.

```
Route::get('/users', function (Request $request) {  
    return $request->input('name');  
});
```

The difference between 'input' and 'query' is that query only works on uri query parameters whereas 'input' works on form fields as well as query parameters.

```
Route::get('/users', function (Request $request) {  
    return $request->input('name', 'Default Name');  
});
```

With 'input' I can also include a default value, so if the value is not in the uri or the form input then the default will be returned.

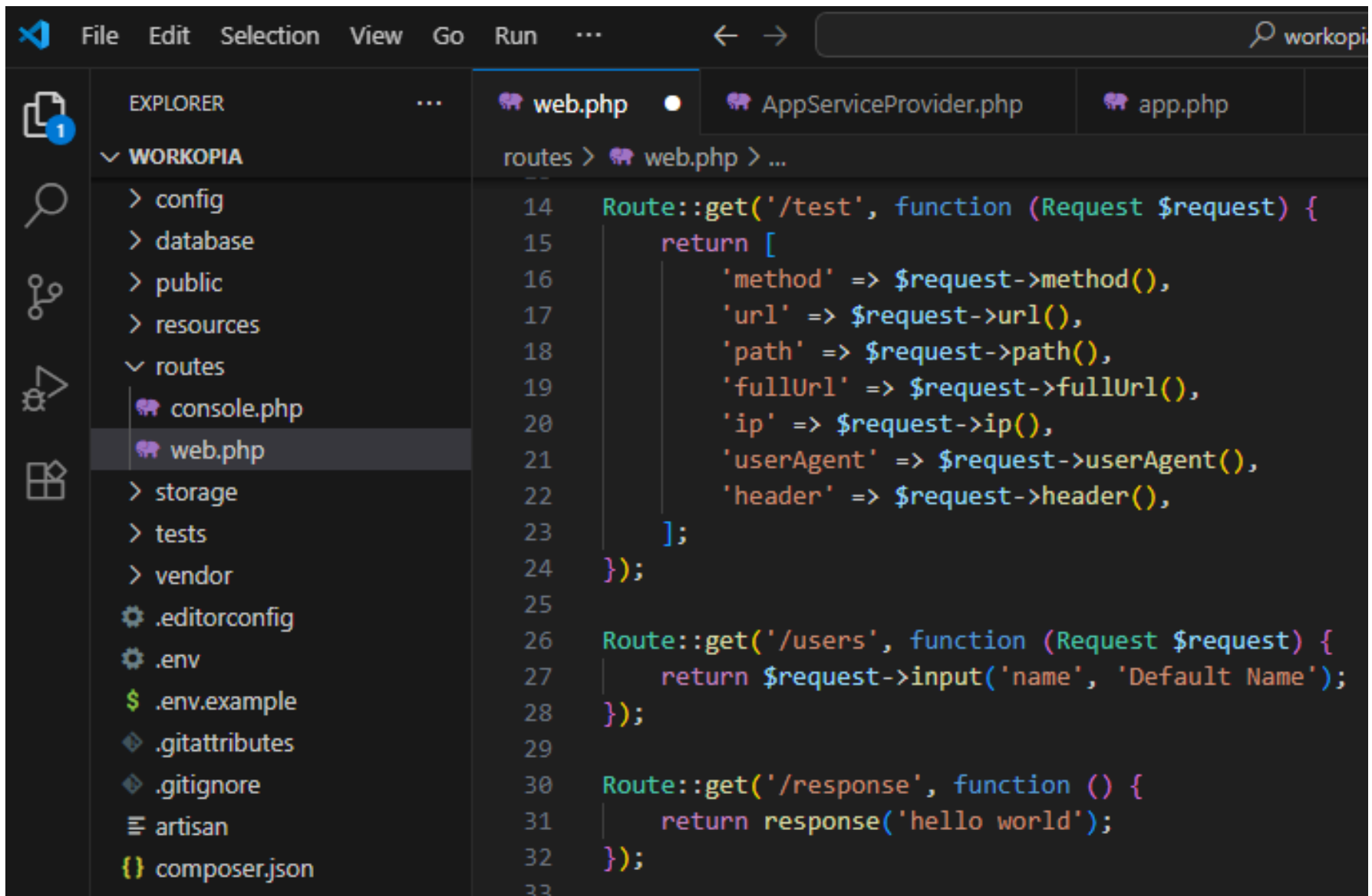


In the terminal of vscode, I can use herd secure command at the project root and this will restart NGINX so that the site is self signing for https.

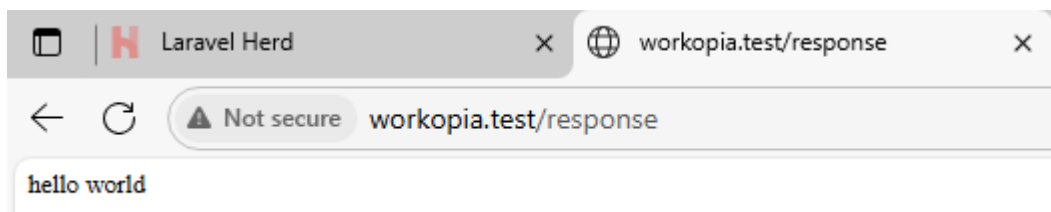
2.12 Response Helper

The response helper can modify the http response headers to change the following parameters in the response object:

1. Set status code
2. Set headers
3. Response body
4. Set cookies
5. Set Session Values
6. Download Files
7. Stream Data
8. Cache Control
9. JSON & JSONP



```
14 Route::get('/test', function (Request $request) {
15     return [
16         'method' => $request->method(),
17         'url' => $request->url(),
18         'path' => $request->path(),
19         'fullUrl' => $request->fullUrl(),
20         'ip' => $request->ip(),
21         'userAgent' => $request->userAgent(),
22         'header' => $request->header(),
23     ];
24 });
25
26 Route::get('/users', function (Request $request) {
27     return $request->input('name', 'Default Name');
28 });
29
30 Route::get('/response', function () {
31     return response('hello world');
32 });
33
```



The Response helper is available globally without specifying any additional lines of code.

2.13 Response Header Http Code

```
Route::get('notfound', function () {  
    return response('page Not Found', 404);  
});
```

I can also include the http code in the response helper and it will set this automatically in the http response.

The screenshot shows a web browser with the address bar displaying 'workopia.test/notfound'. The page content shows 'page Not Found'. The Chrome DevTools Network tab is open, showing a list of network requests. The first request, 'notfound', has a status of 404 and a type of 'document'. The second request, 'favicon.ico', has a status of 200 and a type of 'x-icon'.

Name	Status	Type	Init
notfound	404	document	Oth
favicon.ico	200	x-icon	Oth

2.14 Response Header Content Type

```
Route::get('/response', function () {  
    return response('<h1>hello world</h1>', 200)->header('content-Type', 'text/plain');  
});
```

I can also specify the content type in the response. For example, text/plain will return as text only and not render html markup

The screenshot shows a web browser with the URL `workopia.test/response`. The browser's developer tools are open, displaying the Network tab. A list of requests is shown, with the 'response' request selected. The 'Headers' sub-tab is active, showing the 'Response Headers' section. The 'Content-Type' header is set to 'text/plain; charset=UTF-8'. The browser's address bar shows the URL and a 'Not secure' warning. The page content displays the text '<h1>hello world</h1>'. The browser's tabs show 'Laravel Herd' and 'workopia.test/response'.

Name	Value
Cache-Control	no-cache, private
Connection	keep-alive
Content-Encoding	gzip
Content-Type	text/plain; charset=UTF-8
Date	Thu, 19 Jun 2025 22:11:45 GMT


```
Route::get('/response', function () {  
    return response('<h1>hello world</h1>', 200)->header('content-Type', 'text/html');  
});
```

Laravel Herd

workopia.test/response

Not secure workopia.test/response

hello world

I can also specify the
content type as
text/html

Network tab interface showing response details:

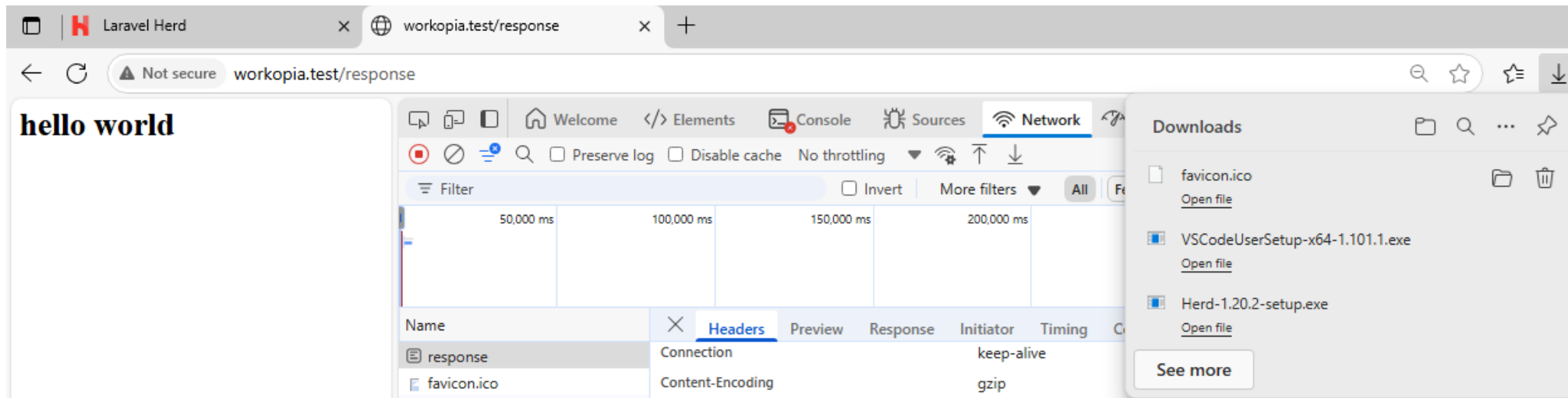
- Timeline: 100 ms, 200 ms, 300 ms, 400 ms, 500 ms, 600 ms
- Filter: [Filter] [Invert] [More filters]
- Response list:
 - response
 - favicon.ico
- Headers tab selected:

Name	Value
Connection	keep-alive
Content-Encoding	gzip
Content-Type	text/html; char:
Date	Thu, 19 Jun 202
Server	nginx/1.25.2
Set-Cookie	XSRF-

2.15 Response Header Download

```
Route::get('/download', function () {  
    return response()->download(public_path('favicon.ico'));  
});
```

I can send a download using the response header.



The screenshot shows a web browser at `workopia.test/response` displaying "hello world". The developer tools are open, showing the Network tab. The 'response' request is selected, and its headers are visible, showing 'Content-Encoding: gzip'. A 'Downloads' panel is also open, showing the downloaded files: `favicon.ico`, `VSCodeUserSetup-x64-1.101.1.exe`, and `Herd-1.20.2-setup.exe`.

Name	Headers	Preview	Response	Initiator	Timing
response	Connection			keep-alive	
favicon.ico	Content-Encoding			gzip	

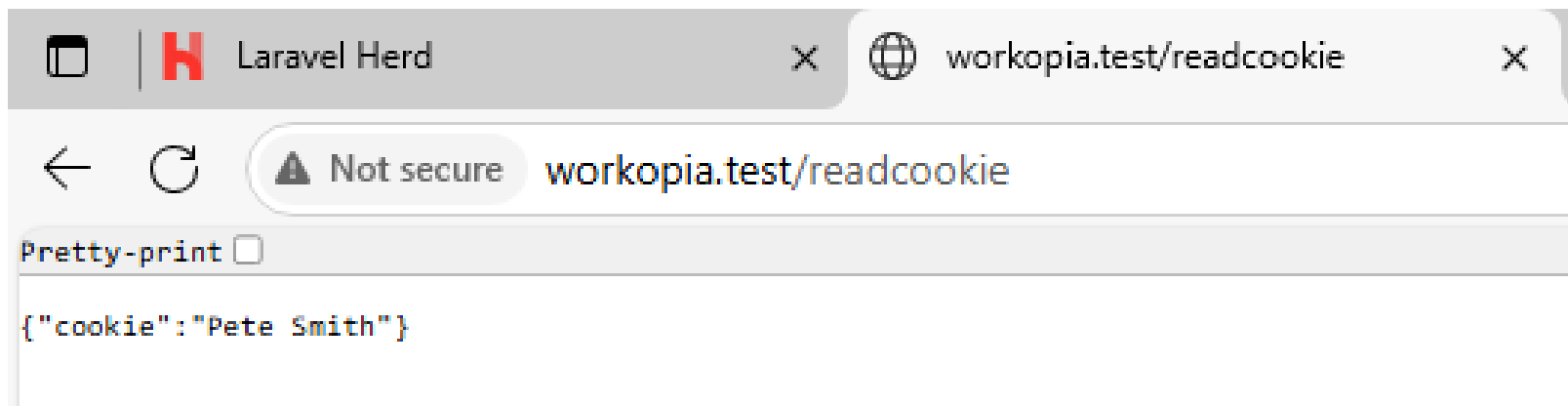
2.15 Response Header Cookie

```
Route::get('/cookie', function () {
    return response()->json(['name' => 'John Doe'])->cookie('name', 'John Doe');
});
```

Here I am explicitly specifying the response as JSON and also setting a cookie. Note how the cookie is set as a key value pair and in the browser the value is encrypted by default by Laravel.

[illegible]

```
Route::get('/cookie', function () {  
    return response()->json(['name' => 'john Doe'])->cookie('name', 'Pete Smith');  
});  
  
Route::get('/readcookie', function (Request $request) {  
    $cookieValue = $request->cookie('name');  
    return response()->json(['cookie' => $cookieValue]);  
});
```



I can get the cookie from the browser using the request cookie inbuilt function.

3. Views & Controllers

[2.1 Create & Display Views](#)

[3.2 Passing Data to views](#)

[3.3 Blade Templates & Directives](#)

[3.4 More Loop Variables & \\$loop Variable](#)

[3.5 Create Controllers](#)

[3.6 Params & Requests in Controller](#)

[3.7 Generate Resource Routes & Methods](#)

[3.8 Type Hinting in Controllers](#)

[3.9 Layouts with Template Inheritance](#)

[3.10 Partial & @include directive](#)

3.1 Create & Display Views

```
<!DOCTYPE html>
<html lang="en">
```

In the views folder I have created a jobs.php file with some boilerplate html.

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Job Listings</title>
</head>
```

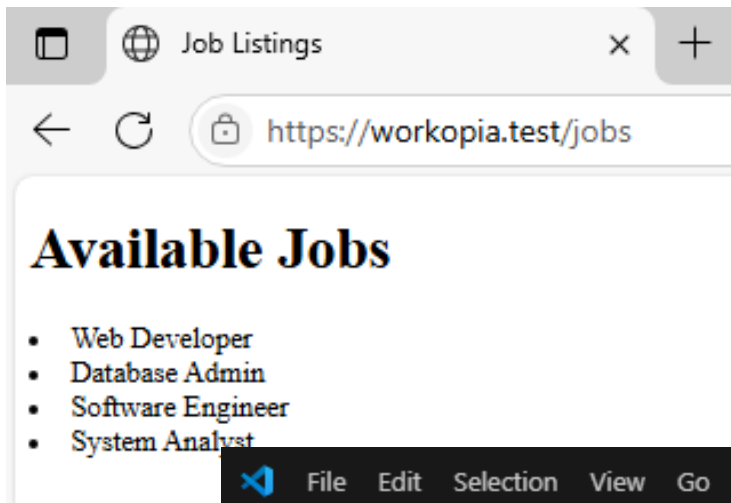
```
<body>
  <h1>Available Jobs</h1>
  <li>Web Developer</li>
  <li>Database Admin</li>
  <li>Software Engineer</li>
  <li>System Analyst</li>
</body>
</html>
```

```
<?php
use Illuminate\Support\Facades\Route;

Route::get('/', function () {
    return view('welcome');
});

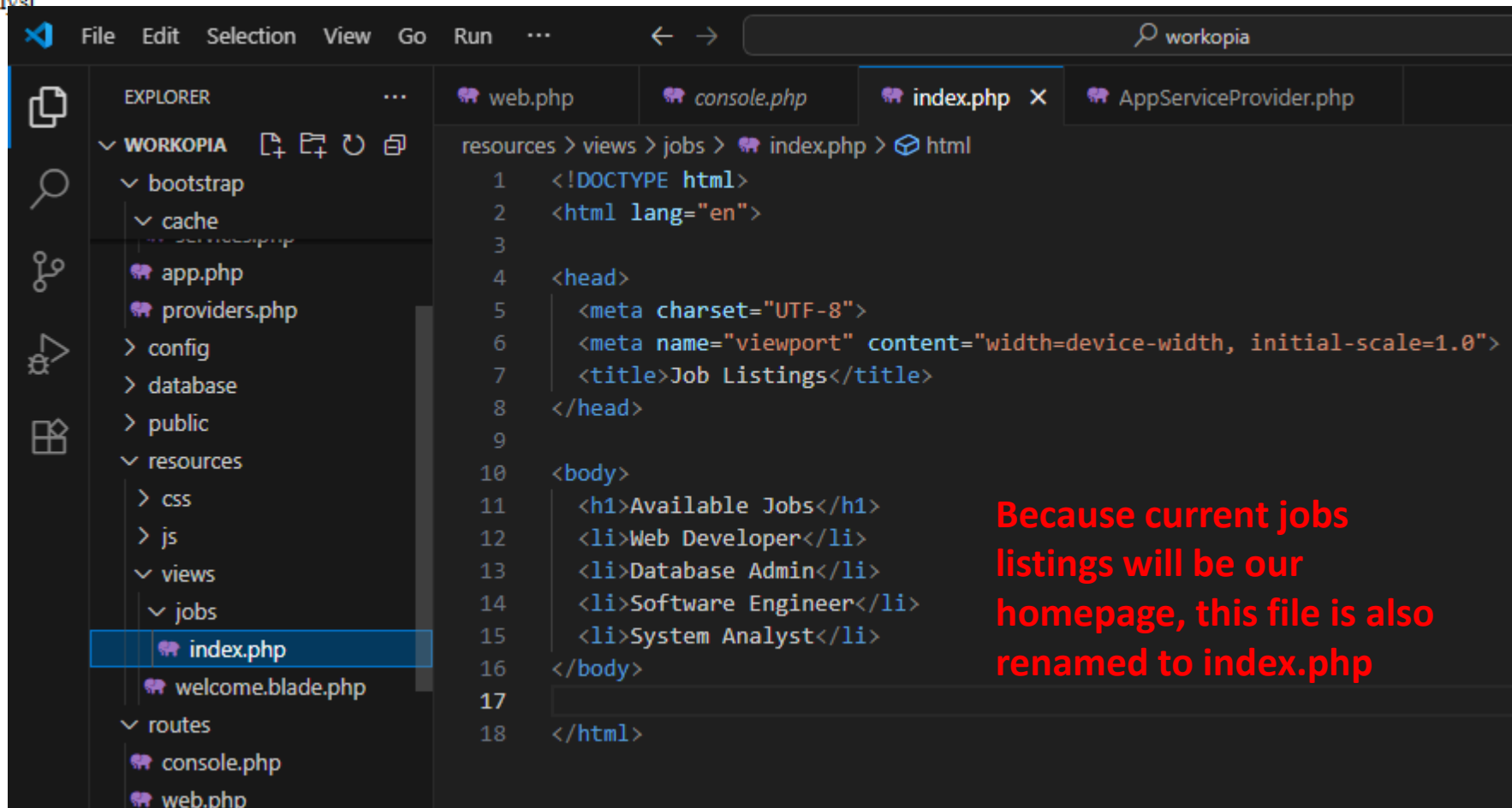
Route::get('jobs', function () {
    return view('jobs');
})->name('jobs');
```

In the routes web.php file I have created a route pointing to the views/jobs.php file

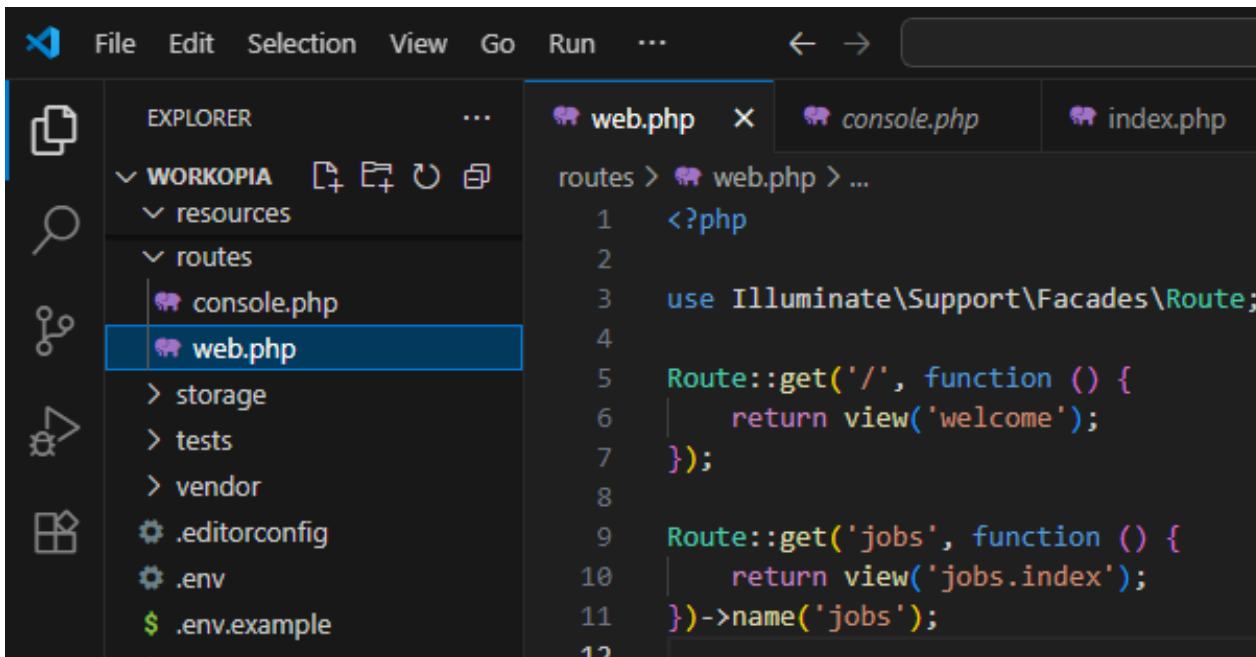


Now the '/jobs' end point lands on the html page created previously.

To have structured and organised code it is worth creating sub folders in the view folder. In this project there will be multiple files relating to jobs



Because current jobs listings will be our homepage, this file is also renamed to index.php

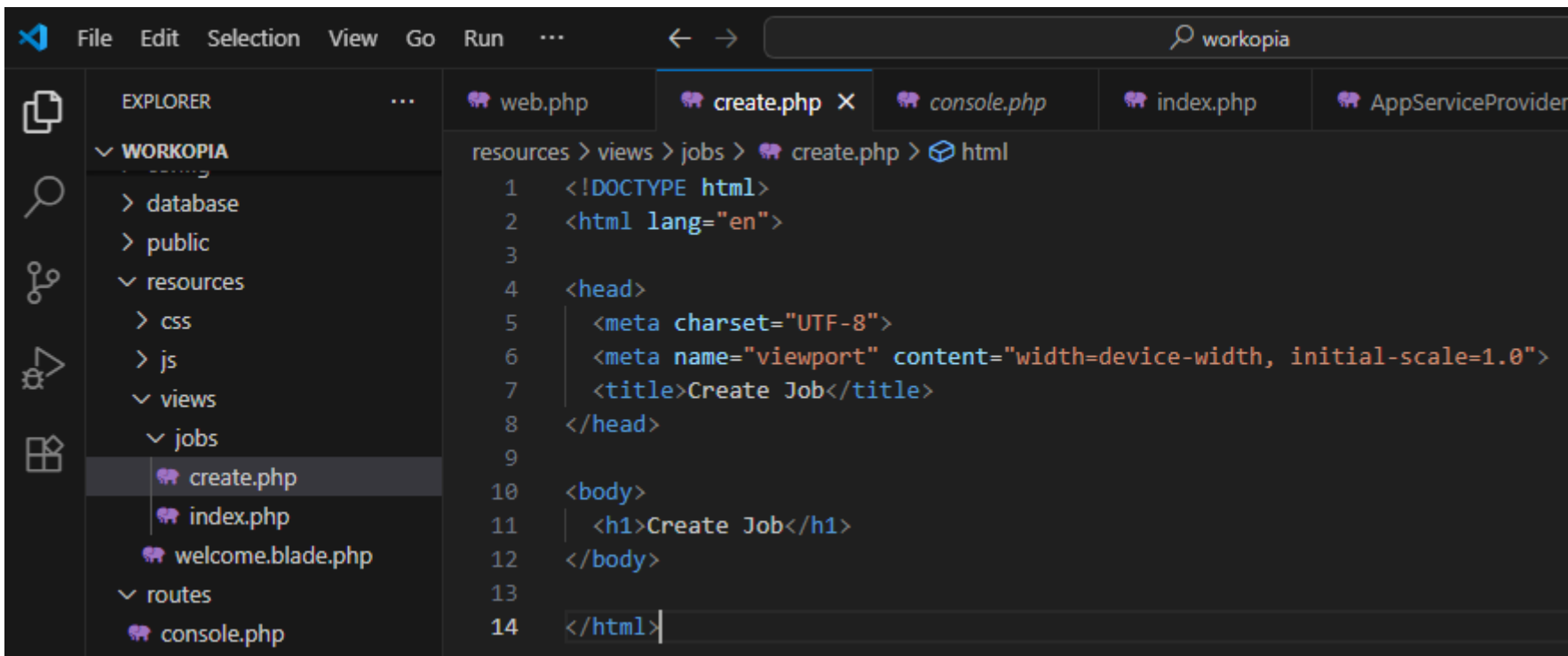


This screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left displaying the project structure. The 'resources' folder is expanded, showing 'console.php' and 'web.php', with 'web.php' selected. The main editor displays the contents of 'web.php', which defines two routes: a root route ('/') returning 'welcome' and a route for '/jobs' returning 'jobs.index'.

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/', function () {
6     return view('welcome');
7 });
8
9 Route::get('jobs', function () {
10     return view('jobs.index');
11 }->name('jobs');
12
```

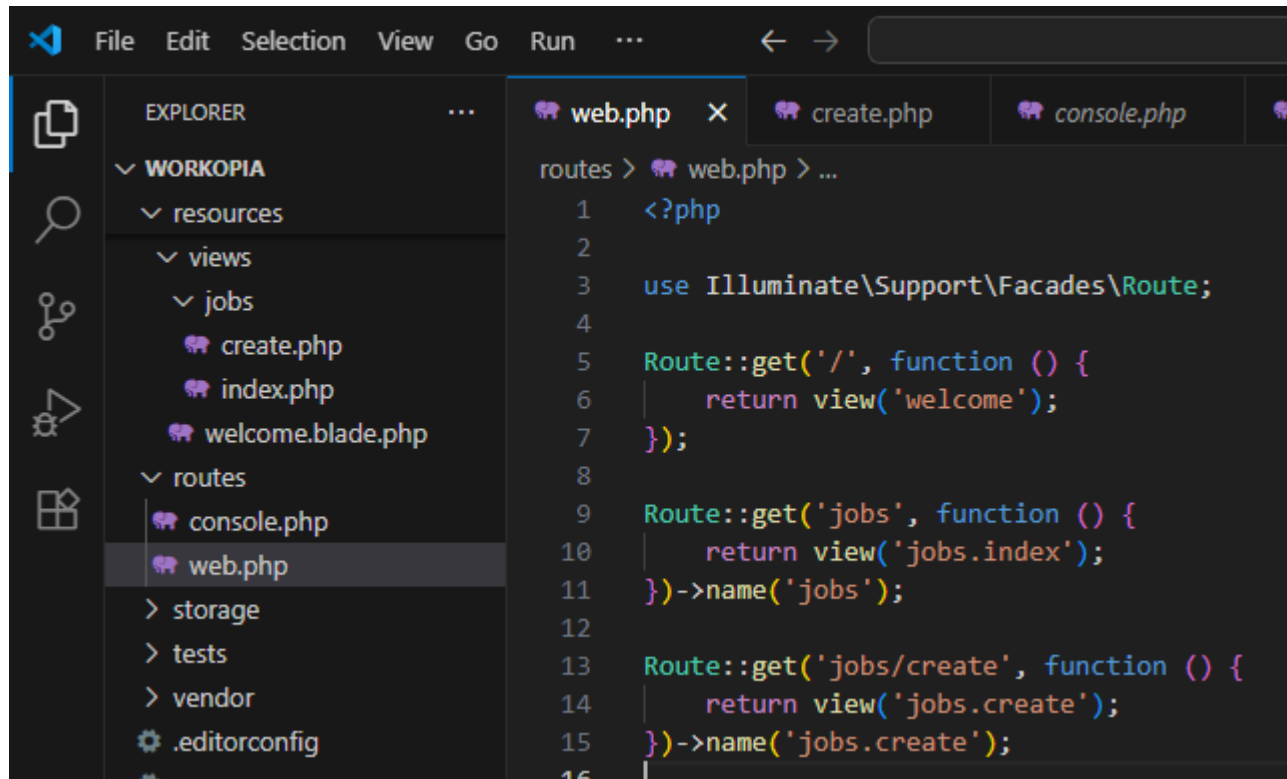
The web.php routing file is also modified to reflex the new name of the homepage.

I also make within the view/jobs/ folder another html file to create a job posting.



This screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left. The 'views' folder is expanded, showing a 'jobs' subfolder containing 'create.php' and 'index.php', with 'create.php' selected. The main editor displays the HTML content of 'create.php', which includes a basic HTML structure with a title 'Create Job'.

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5     <meta charset="UTF-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title>Create Job</title>
8 </head>
9
10 <body>
11     <h1>Create Job</h1>
12 </body>
13
14 </html>
```

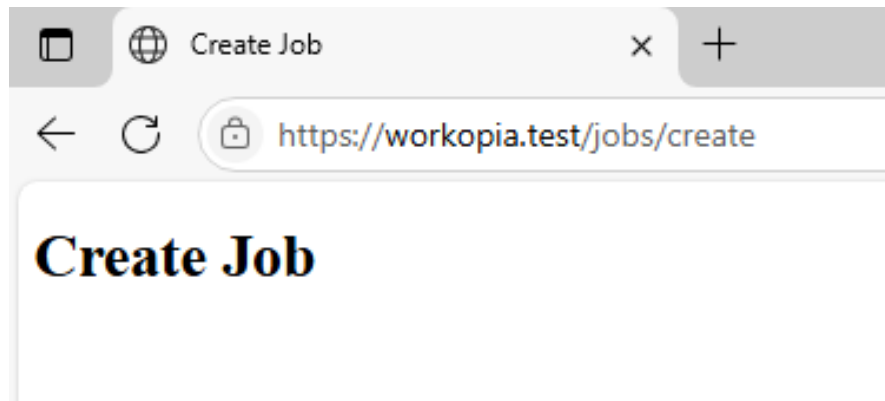



The screenshot shows the Visual Studio Code editor with the 'web.php' file open in the 'routes' directory. The file contains the following PHP code:

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/', function () {
6     return view('welcome');
7 });
8
9 Route::get('jobs', function () {
10     return view('jobs.index');
11 }->name('jobs');
12
13 Route::get('jobs/create', function () {
14     return view('jobs.create');
15 }->name('jobs.create');
```

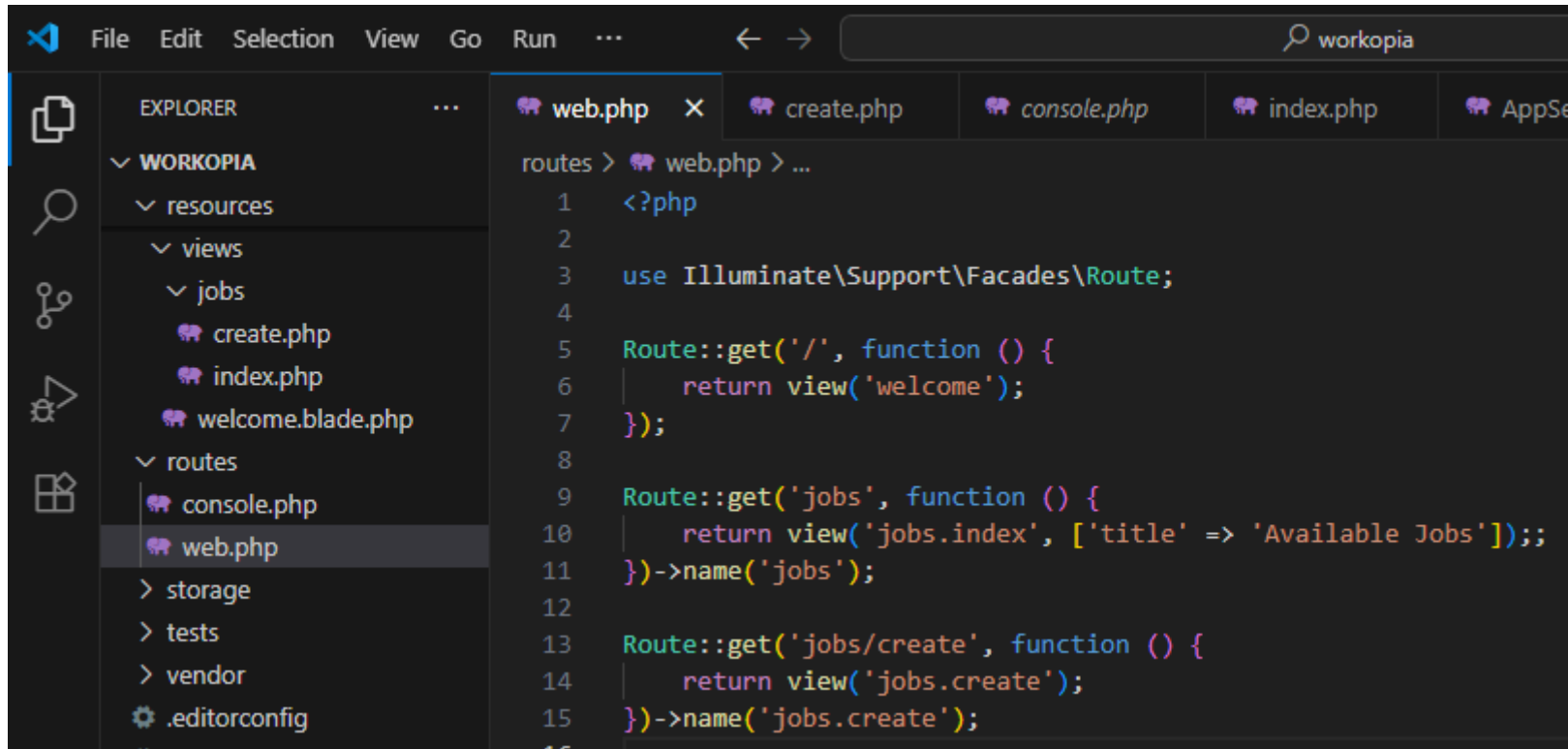
The Explorer sidebar on the left shows the project structure with 'resources' > 'views' > 'jobs' > 'web.php' selected.

Adding in the routing to the /jobs/create endpoint



And verify that it works in the browser.

3.2 Passing Data to Views

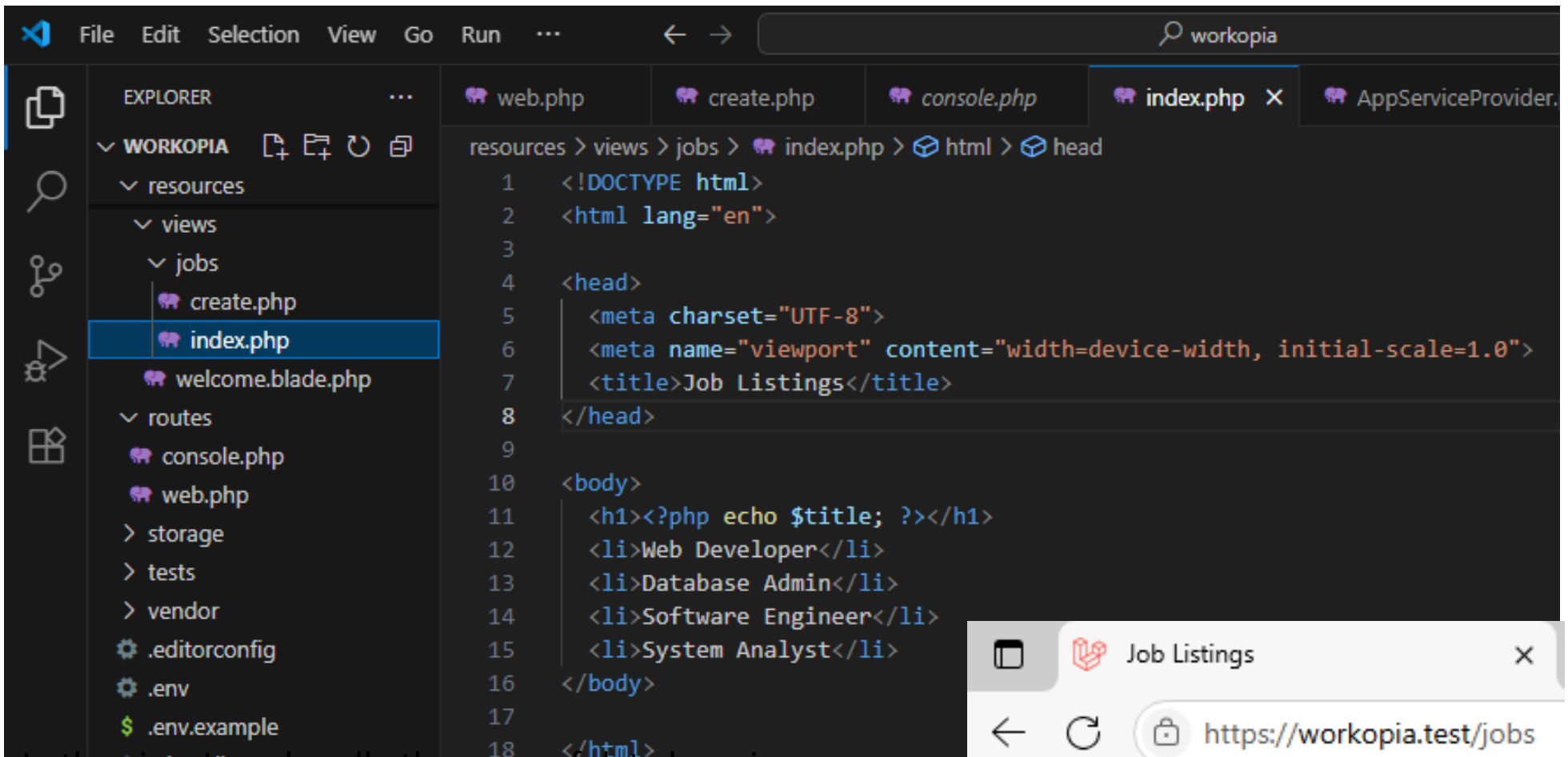


```
File Edit Selection View Go Run ... workopia

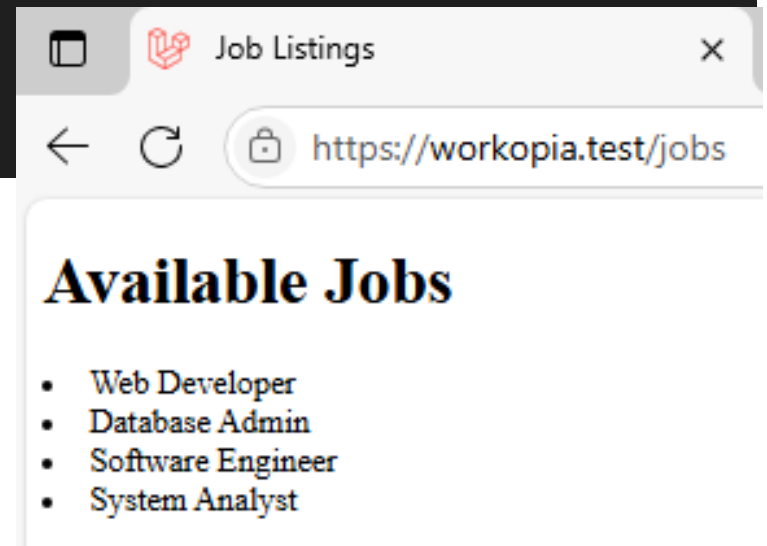
EXPLORER
WORKOPIA
  resources
    views
      jobs
        create.php
        index.php
        welcome.blade.php
    routes
      console.php
      web.php
  storage
  tests
  vendor
  .editorconfig

routes > web.php > ...
1  <?php
2
3  use Illuminate\Support\Facades\Route;
4
5  Route::get('/', function () {
6      return view('welcome');
7  });
8
9  Route::get('jobs', function () {
10     return view('jobs.index', ['title' => 'Available Jobs']);
11 });->name('jobs');
12
13 Route::get('jobs/create', function () {
14     return view('jobs.create');
15 });->name('jobs.create');
```

I can pass an array of data into the view from the routing web.php file when I call the route.



In the view I can handle the array of data by using php opening and closing tags to target each element of the array and insert it into the html.



File Edit Selection View Go Run ...

workopia

EXPLORER

WORKOPIA

resources

views

jobs

create.php

index.php

welcome.blade.php

routes

console.php

web.php

storage

tests

vendor

.editorconfig

web.php

create.php

console.php

index.php

AppSer

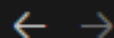
routes > web.php > ...

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/', function () {
6     return view('welcome');
7 });
8
9 Route::get('jobs', function () {
10    return view('jobs.index')->with('title', 'Available Jobs');
11 }->name('jobs');
12
13 Route::get('jobs/create', function () {
14    return view('jobs.create');
15 }->name('jobs.create');
```

The with method can also be used to pass an array of data.



File Edit Selection View Go Run ...



workopia



EXPLORER



web.php X

create.php

console.php

index.php

A



WORKOPIA

resources

views

jobs

create.php

index.php

welcome.blade.php



routes

console.php

web.php

> storage

> tests

> vendor

.editorconfig

.env

.env.example

.gitattributes

.gitignore

.rnd

artisan

routes > web.php > ...

1 <?php

2

3 use Illuminate\Support\Facades\Route;

4

5 Route::get('/', function () {

6 | return view('welcome');

7 });

8

9 Route::get('jobs', function () {

10 | \$title = 'Available Jobs';

11 | \$jobs = [

12 | 'Web Developer',

13 | 'Database Admin',

14 | 'Software Engineer',

15 | 'System Analyst'

16 |];

17 | return view('jobs.index', compact('title', 'jobs'));

18 | }->name('jobs');

19

20 Route::get('jobs/create', function () {

21 | return view('jobs.create');

22 | }->name('jobs.create');

23

A cleaner method is
to use compact to
pass string and array
variables in to the
view

File Edit Selection View Go Run ...workopia

EXPLORERWORKOPIAresourcesviewsjobscreate.phpindex.phpwelcome.blade.phproutesconsole.phpweb.phpstoragetestsvendor.editorconfig.env.env.example.gitattributes.gitignore

resources > views > jobs > index.php > html

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Job Listings</title>
8 </head>
9
10 <body>
11   <h1><?php echo $title; ?></h1>
12   <ul>
13     <?php foreach ($jobs as $job) : ?>
14       <li><?php echo $job; ?></li>
15     <?php endforeach; ?>
16   </ul>
17 </body>
18
19 </html>
```

Now I can iterate over the jobs array within a foreach loop. Note that it is best to use 'htmlspecialchars' to prevent SQL injection.

<?php echo htmlspecialchars(\$job, ENT_QUOTES, 'UTF-8'); ?>

3.3 Blade Templates & Directives

File Edit Selection View Go Run ... workopia

EXPLORER

- WORKOPIA
 - resources
 - views
 - jobs
 - create.blade.php
 - index.blade.php**
 - welcome.blade.php
 - routes
 - console.php
 - web.php
 - storage
 - tests
 - vendor
 - .editorconfig
 - .env
 - .env.example
 - .gitattributes
 - .gitignore

resources > views > jobs > index.blade.php > html

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>{{ $title }}</title>
8 </head>
9
10 <body>
11   <h1>{{ $title }}</h1>
12   <ul>
13     @foreach($jobs as $job)
14       <li>{{ $job }}</li>
15     @endforeach
16   </ul>
17 </body>
18
19 </html>
```

To convert the views to blade templates I rename them adding '.blade'

To use a string variable, I wrap it in double curly quotes. Everything else is handled by blade.

I can use an @ directive to signify loops.

Blade templates are cleaner and do all the heavy lifting

The screenshot shows a code editor with a sidebar on the left containing a file explorer. The main editor area displays a Blade template file named `index.blade.php`. The code is a standard HTML structure with a head section and a body section. The body section contains a conditional directive `@if(!empty($jobs))` followed by a loop `@foreach($jobs as $job)` that renders a list of jobs. If the jobs array is empty, the `@else` block renders a paragraph stating "No Jobs available". The code is annotated with red text and arrows explaining the directives.

File Explorer (Left Sidebar):

- WORKOPIA
 - resources
 - views
 - jobs
 - create.blade.php
 - index.blade.php**
 - welcome.blade.php
 - routes
 - console.php
 - web.php
 - storage
 - tests
 - vendor
 - .editorconfig
 - .env
 - .env.example
 - .gitattributes
 - .gitignore
 - .rnd
 - artisan
 - composer.json

Code Editor (Main Area):

```
resources > views > jobs > index.blade.php > html > body > ul
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>{{$title}}</title>
8 </head>
9
10 <body>
11   <h1>{{$title}}</h1>
12   @if(!empty($jobs))
13     <ul>
14       @foreach($jobs as $job)
15         <li>{{$job}}</li>
16       @endforeach
17     </ul>
18   @else
19     <p>No Jobs available</p>
20   @endif
21 </body>
22
23 </html>
```

Annotations (Red Text and Arrows):

- I can use an @If directive to check if the Jobs array is not empty.** (Arrow points to line 12: `@if(!empty($jobs))`)
- If there are jobs then it will render the UL** (Arrow points to line 14: `@foreach($jobs as $job)`)
- @else will render a paragraph if the jobs array is empty.** (Arrow points to line 18: `@else`)
- Always remember to close conditionals and loops with an @end..** (Arrow points to line 20: `@endif`)

FileEditSelectionViewGoRun...workopia

EXPLORERWORKOPIAresourcesviewsjobscreate.blade.phpindex.blade.phpwelcome.blade.phproutesconsole.phpweb.phpstoragetestsvendor.editorconfig.env.env.example.gitattributes.gitignore.rndartisan

web.phpcreate.blade.phpconsole.phpindex.blade.php

routes > web.php > ...

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/', function () {
6     return view('welcome');
7 });
8
9 Route::get('jobs', function () {
10    $title = 'Available Jobs';
11    $jobs = [
12        // 'Web Developer',
13        // 'Database Admin',
14        // 'Software Engineer',
15        // 'System Analyst'
16    ];
17    return view('jobs.index',
18    )->name('jobs');
19
20 Route::get('jobs/create', func
21     return view('jobs.create')
22 )->name('jobs.create');
23
```

Testing the conditional directive by removing jobs from the array

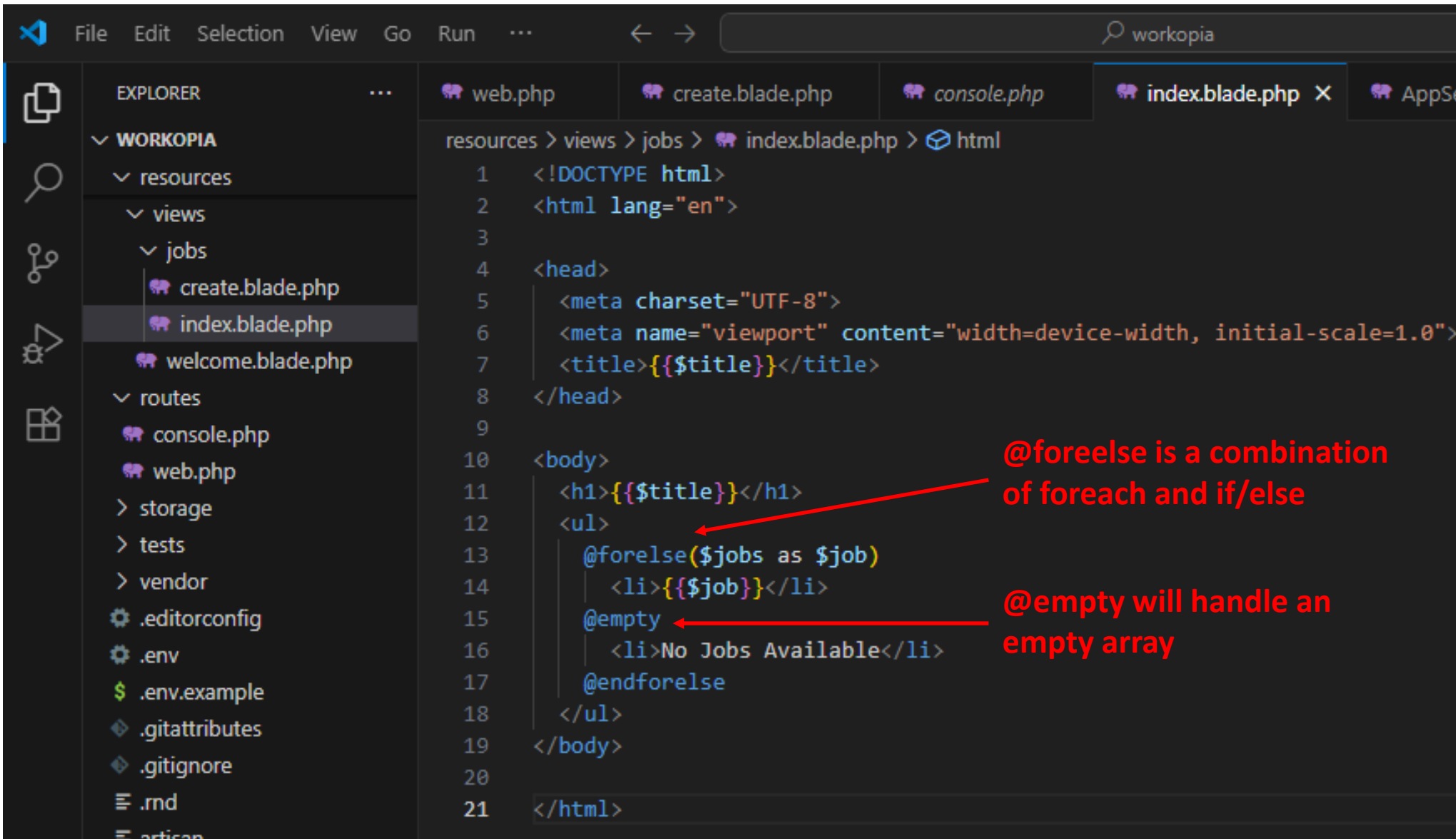
Available Jobs

https://workopia.test/jobs

Available Jobs

No Jobs available

3.4 More loop variables & \$loop Variable



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a 'resources' folder containing 'views' and 'jobs' subfolders. The 'jobs' folder contains 'create.blade.php' and 'index.blade.php'. The 'index.blade.php' file is selected and its content is displayed in the code editor. The code is a Blade template for a job listing page. It includes a DOCTYPE declaration, a language attribute, a head section with meta tags for charset and viewport, and a body section with an h1 tag and a list of jobs. The list is generated using the `@forelse` directive, which is annotated with a red arrow and text stating it is a combination of `foreach` and `if/else`. The `@empty` directive is also annotated with a red arrow and text stating it will handle an empty array. The `@endforelse` directive closes the loop.

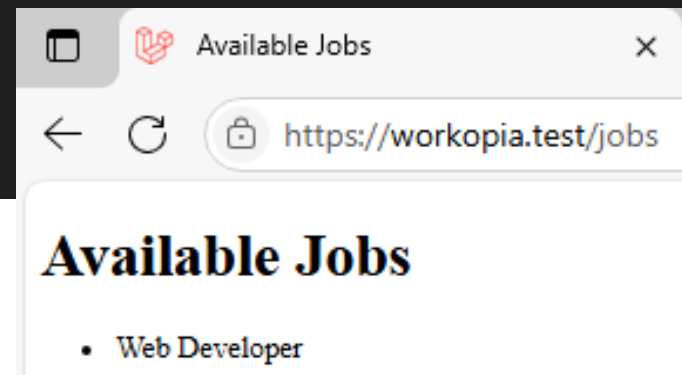
```
resources > views > jobs > index.blade.php > html
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>{{ $title }}</title>
8 </head>
9
10 <body>
11   <h1>{{ $title }}</h1>
12   <ul>
13     @forelse($jobs as $job)
14       <li>{{ $job }}</li>
15     @empty
16       <li>No Jobs Available</li>
17     @endforelse
18   </ul>
19 </body>
20
21 </html>
```

@forelse is a combination of foreach and if/else

@empty will handle an empty array

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>{{$title}}</title>
8 </head>
9
10 <body>
11   <h1>{{$title}}</h1>
12   <ul>
13     @forelse($jobs as $job)
14       @if($job == 'Database Admin')
15         @break
16       @endif
17       <li>{{$job}}</li>
18     @empty
19       <li>No Jobs Available</li>
20     @endforelse
21   </ul>
22 </body>
```

An @if and @break can be used to abort the loop if the job is 'database admin'



FileEditSelectionViewGoRun...<=>workopia

EXPLORERWORKOPIAresourcesviewsjobscreate.blade.phpindex.blade.phpwelcome.blade.phroutesconsole.phpweb.phpstoragetestsvendor.editorconfig.env.env.example.gitattributes.gitignore.rndartisan

resources > views > jobs > index.blade.php > html > body > ul

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>{{$title}}</title>
8 </head>
9
10 <body>
11   <h1>{{$title}}</h1>
12   <ul>
13     @forelse($jobs as $job)
14       @if($job == 'Database Admin')
15         @continue
16       @endif
17       <li>{{$job}}</li>
18     @empty
19       <li>No Jobs Available</li>
20     @endforelse
21   </ul>
22 </body>
```

@continue will cause it to skip over the 'Database Admin' job.

Available Jobs

https://workopia.test/jobs

Available Jobs

- Web Developer
- Software Engineer
- System Analyst

Available Jobs

https://workopia.test/jobs

Available Jobs

- Web Developer
- Software Engineer
- System Analyst

The screenshot shows the Visual Studio Code editor with a project named 'workopia'. The Explorer sidebar on the left shows the file structure: `WORKOPIA` > `resources` > `views` > `jobs`, where `index.blade.php` is selected. The main editor displays the content of `index.blade.php`, which is an HTML template. It includes a DOCTYPE declaration, a head section with meta tags for charset and viewport, and a body section. The body contains an `h1` tag with a Blade variable `{{ $title }}`, followed by a `ul` tag. Inside the `ul`, there is a `@forelse` loop: `@forelse($jobs as $job)`. The loop body contains an `li` tag with the text `{{ $loop->index }}` followed by a dash and `{{ $job }}`. If there are no jobs, it displays 'No Jobs Available'. The loop ends with `@endforelse`. The file ends with `</html>`. A red arrow points from a text box to the `{{ $loop->index }}` variable in the `li` tag.

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>{{ $title }}</title>
8 </head>
9
10 <body>
11   <h1>{{ $title }}</h1>
12   <ul>
13     @forelse($jobs as $job)
14       <li>{{ $loop->index }} - {{ $job }}</li>
15     @empty
16       <li>No Jobs Available</li>
17     @endforelse
18   </ul>
19 </body>
20
21 </html>
```

The `$loop` variable is a bunch of built in features for loops. Here I am showing the index of the loop

The screenshot shows a web browser window with the title 'Available Jobs'. The address bar shows the URL `https://workopia.test/jobs`. The page content displays the heading 'Available Jobs' followed by a bulleted list of jobs. A red arrow points from the `{{ $loop->index }}` variable in the code to the first item in the list, '0 - Web Developer'.

Available Jobs

- 0 - Web Developer
- 1 - Database Admin
- 2 - Software Engineer
- 3 - System Analyst

File Edit Selection View Go Run ... workopia

EXPLORER

- WORKOPIA
 - resources
 - views
 - jobs
 - create.blade.php
 - index.blade.php**
 - welcome.blade.php
 - routes
 - console.php
 - web.php
 - storage
 - tests
 - vendor
 - .editorconfig
 - .env
 - .env.example
 - .gitattributes
 - .gitignore
 - .rnd
 - artisan

resources > views > jobs > index.blade.php > html

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>{{ $title }}</title>
8 </head>
9
10 <body>
11   <h1>{{ $title }}</h1>
12   <ul>
13     @forelse($jobs as $job)
14       <li>{{ $loop->iteration }} - {{ $job }}</li>
15     @empty
16       <li>No Jobs Available</li>
17     @endforelse
18   </ul>
19 </body>
20
21 </html>
```

The \$loop iteration shows us the number of the item in the array starting from 1.

Available Jobs

https://workopia.test/jobs

Available Jobs

- 1 - Web Developer
- 2 - Database Admin
- 3 - Software Engineer
- 4 - System Analyst

File Edit Selection View Go Run ... workopia

EXPLORER

WORKOPIA

- resources
 - views
 - jobs
 - create.blade.php
 - index.blade.php**
 - welcome.blade.php
- routes
 - console.php
 - web.php
- storage
- tests
- vendor
- .editorconfig
- .env
- .env.example
- .gitattributes
- .gitignore
- .rnd

resources > views > jobs > index.blade.php > html

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>{{ $title }}</title>
8 </head>
9
10 <body>
11   <h1>{{ $title }}</h1>
12   <ul>
13     @forelse($jobs as $job)
14       <li>{{ $loop->remaining }} - {{ $job }}</li>
15     @empty
16       <li>No Jobs Available</li>
17     @endforelse
18   </ul>
19 </body>
20
21 </html>
```

The \$loop remaining shows the number of remaining items in the array.

Available Jobs

https://workopia.test/jobs

Available Jobs

- 3 - Web Developer
- 2 - Database Admin
- 1 - Software Engineer
- 0 - System Analyst

File Edit Selection View Go Run ... workopia

EXPLORER

- WORKOPIA
 - resources
 - views
 - jobs
 - create.blade.php
 - index.blade.php
 - welcome.blade.php
 - routes
 - console.php
 - web.php
 - storage
 - tests
 - vendor
 - .editorconfig
 - .env
 - .env.example
 - .gitattributes
 - .gitignore
 - .rnd

resources > views > jobs > index.blade.php > html

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>{{ $title }}</title>
8 </head>
9
10 <body>
11   <h1>{{ $title }}</h1>
12   <ul>
13     @forelse($jobs as $job)
14       <li>{{ $loop->count }} - {{ $job }}</li>
15     @empty
16       <li>No Jobs Available</li>
17     @endforelse
18   </ul>
19 </body>
20
21 </html>
```

The \$loop count

Available Jobs

https://workopia.test/jobs

Available Jobs

- 4 - Web Developer
- 4 - Database Admin
- 4 - Software Engineer
- 4 - System Analyst

File Edit Selection View Go Run ... workopia

EXPLORER

- WORKOPIA
 - resources
 - views
 - jobs
 - create.blade.php
 - index.blade.php**
 - welcome.blade.php
 - routes
 - console.php
 - web.php
 - storage
 - tests
 - vendor
 - .editorconfig
 - .env
 - .env.example
 - .gitattributes
 - .gitignore
 - .rnd
 - artisan
 - composer.json

resources > views > jobs > index.blade.php > html

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>{{ $title }}</title>
8 </head>
9
10 <body>
11   <h1>{{ $title }}</h1>
12   <ul>
13     @forelse($jobs as $job)
14       @if($loop->first)
15         <li>First: {{ $job }}</li>
16       @else
17         <li>{{ $job }}</li>
18       @endif
19     @empty
20       <li>No Jobs Available</li>
21     @endforelse
22   </ul>
23 </body>
```

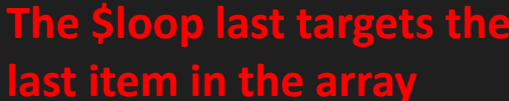
The \$loop first targets the first item in the array

Available Jobs

https://workopia.test/jobs

Available Jobs

- First: Web Developer
- Database Admin
- Software Engineer
- System Analyst



File Edit Selection View Go Run ...

workopia

EXPLORER

WORKOPIA

resources

views

jobs

create.blade.php

index.blade.php

welcome.blade.php

routes

console.php

web.php

storage

tests

vendor

.editorconfig

.env

.env.example

.gitattributes

.gitignore

.rnd

artisan

composer.json

web.php

create.blade.php

console.php

index.blade.php X

AppSer

resources > views > jobs > index.blade.php > html

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>{{ $title }}</title>
8 </head>
9
10 <body>
11   <h1>{{ $title }}</h1>
12   <ul>
13     @foreach($jobs as $job)
14       @if($loop->even)
15         <li>Even: {{ $job }}</li>
16       @else
17         <li>{{ $job }}</li>
18       @endif
19     @empty
20       <li>No Jobs Available</li>
21     @endforeach
22   </ul>
23 </body>
```

The \$loop even targets all even item in the array. Odd targets all odd items

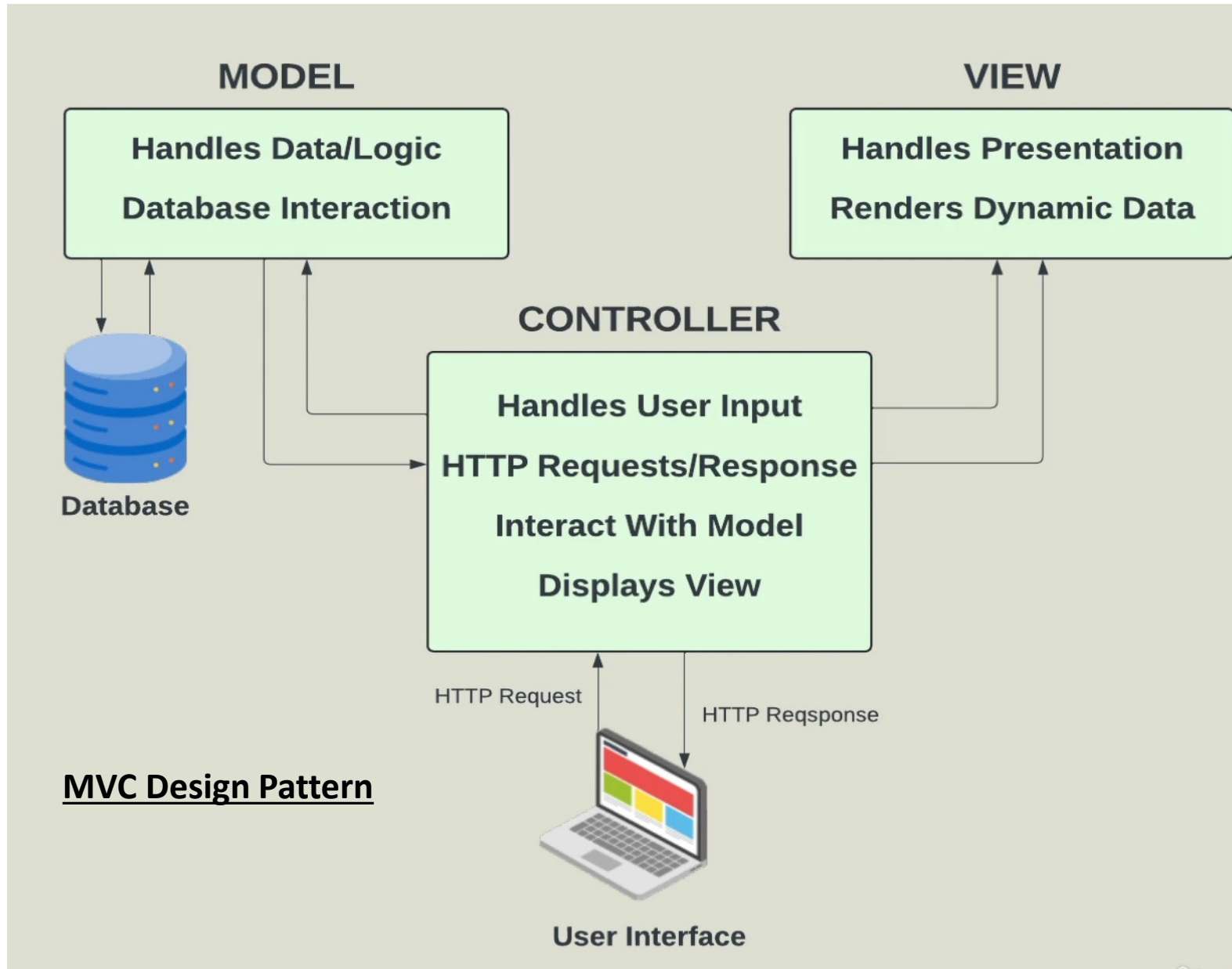
Available Jobs

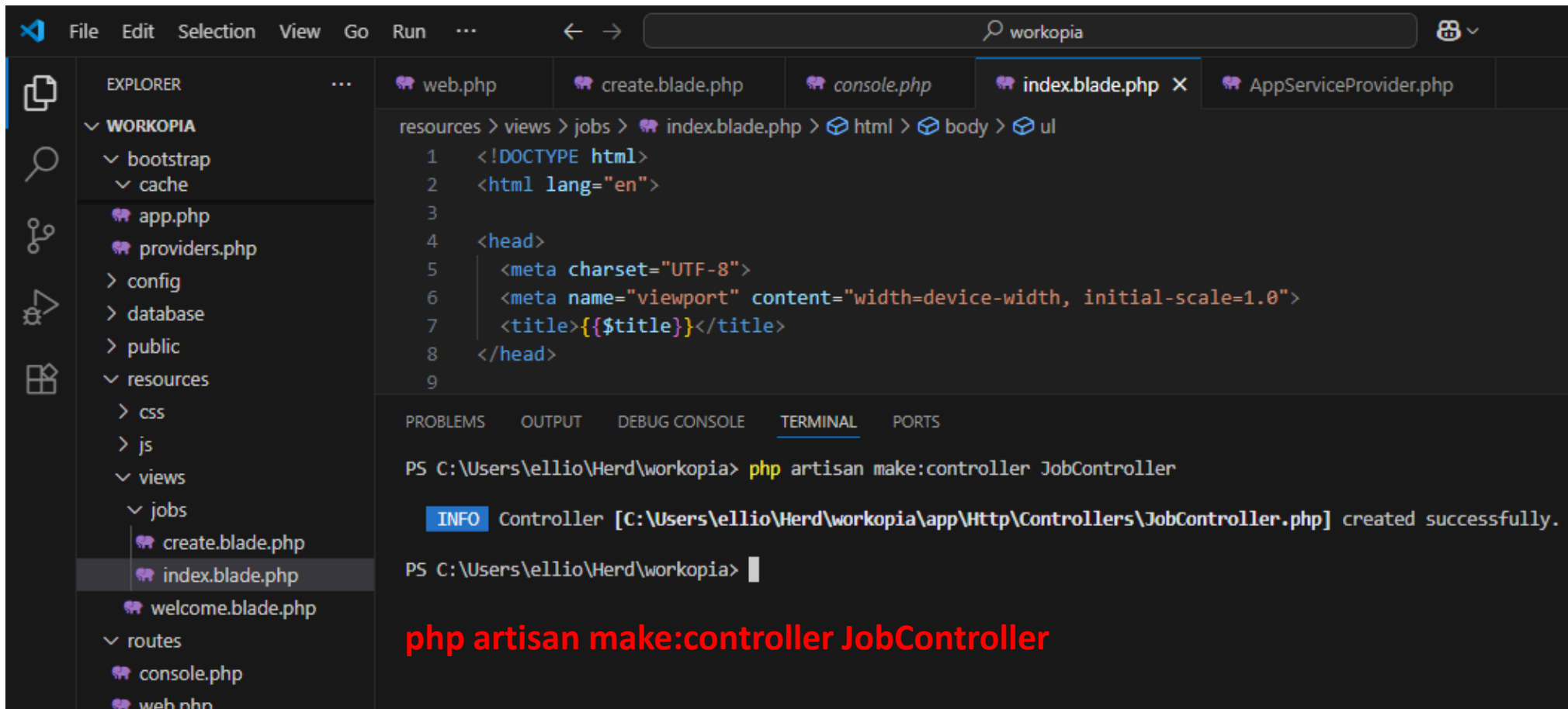
https://workopia.test/jobs

Available Jobs

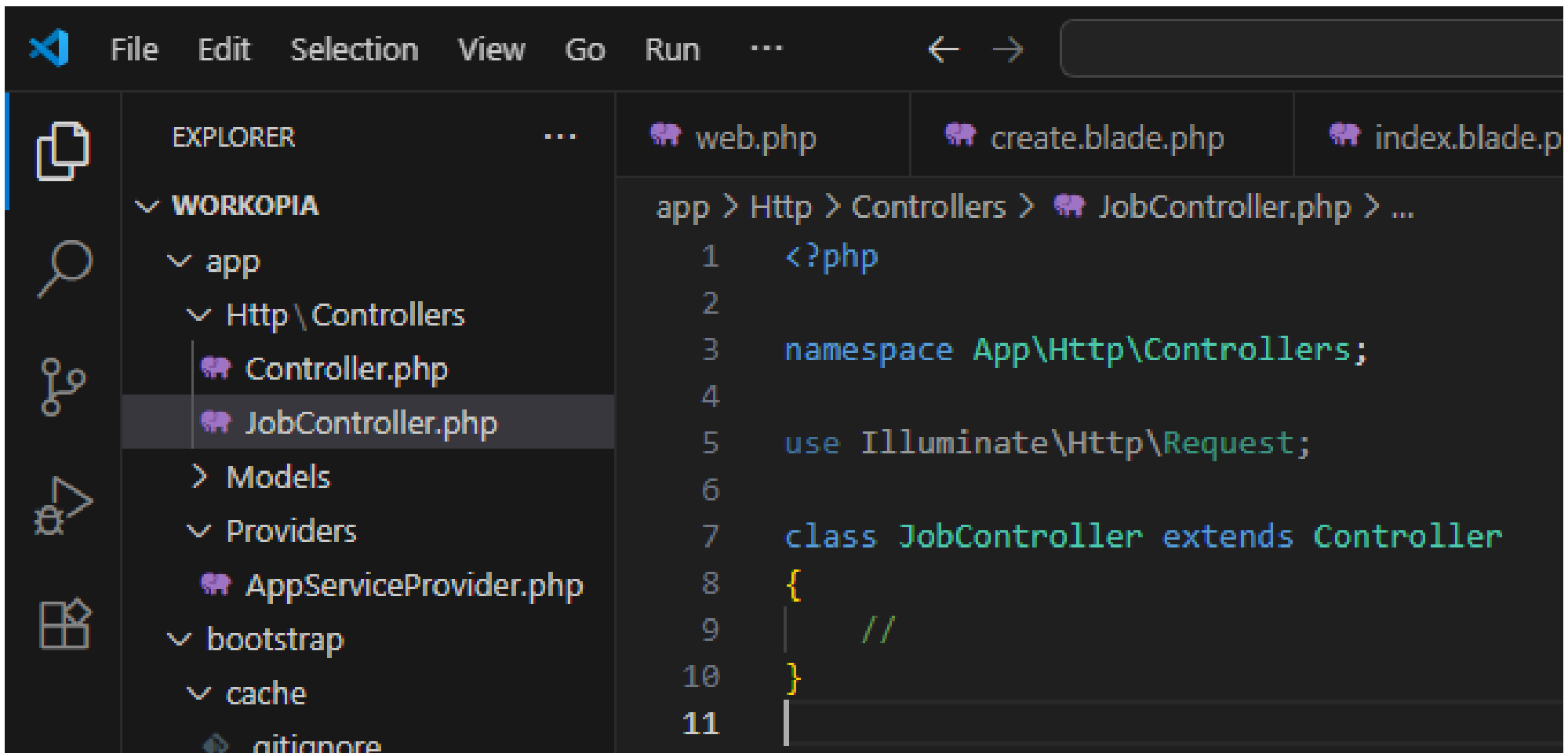
- Web Developer
- Even: Database Admin
- Software Engineer
- Even: System Analyst

3.5 Create Controllers





I can build a controller manually, but it is easier to do it automatically using artisan from the vscode terminal pane.



This builds the Job Controller which is just a php class that extends the controller class. Note that it is named JobControlloer (singular) not plural JobsController.

At this point it is good to understand the Laravel Naming Conventions as laid out here:

<https://github.com/alexymezenin/laravel-best-practices/blob/master/README.md#follow-laravel-naming-conventions>



EXPLORER



web.php

create.blade.php

index.blade.php

JobController.php



WORKOPIA

app

Http\Controllers

Controller.php

JobController.php



> Models

Providers

AppServiceProvider.php



bootstrap

cache

.gitignore

packages.php

services.php

app.php

providers.php

> config

> database

> public

resources

app > Http > Controllers > JobController.php > ...

1 <?php

2

3 namespace App\Http\Controllers;

4

5 use Illuminate\Http\Request;

6

7 class JobController extends Controller

8 {

9

10 public function index()

11 {

12 \$title = 'Available Jobs';

13

14 \$jobs = [

15

16 'Web Developer',

17

18 'Database Admin',

19

20 'Software Engineer',

21

22 'System Analyst'

23

24

25

];

return view('jobs.index', compact('title', 'jobs'));

}

}

I can copy the method from the web.php routing file and dump it into the controller.

The screenshot shows the Visual Studio Code editor with the 'web.php' file open. The Explorer sidebar on the left shows the project structure: WORKOPIA > resources > views > jobs. The 'web.php' file is selected in the 'routes' folder. The code in 'web.php' is as follows:

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\JobController;
5
6 Route::get('/', function () {
7     return view('welcome');
8 });
9
10 Route::get('jobs', [JobController::class, 'index']);
11
12 Route::get('jobs/create', function () {
13     return view('jobs.create');
14 })->name('jobs.create');
```

Annotations with red arrows point to specific parts of the code:

- Use the JobController in the web.php routing** points to the `use App\Http\Controllers\JobController;` line.
- Make the '/jobs' endpoint route to the JobsController** points to the `Route::get('jobs', [JobController::class, 'index']);` line.
- by including the class and the method** points to the `[JobController::class, 'index']` part of the route definition.

The screenshot shows a web browser window with the URL `https://workopia.test/jobs`. The page title is 'Available Jobs'. Below the title, there is a list of job roles:

- Web Developer
- Database Admin
- Software Engineer
- System Analyst

Verify that the jobs page is still loading but now via the 'JobController'



File Edit Selection View Go Run ...



workopia



EXPLORER



WORKOPIA

app

Http\ Controllers

Controller.php

JobController.php

Models

Providers

AppServiceProvider.php

bootstrap

cache

.gitignore

packages.php

services.php

app.php

providers.php

config

database

public

resources

css

js

views

web.php

create.blade.php

index.blade.php

JobController.php

app > Http > Controllers > JobController.php > JobController > create

```
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class JobController extends Controller
8 {
9     public function index()
10     {
11         $title = 'Available Jobs';
12         $jobs = [
13             'Web Developer',
14             'Database Admin',
15             'Software Engineer',
16             'System Analyst'
17         ];
18         return view('jobs.index', compact('title', 'jobs'));
19     }
20
21     public function create()
22     {
23         return view('jobs.create');
24     }
25 }
```

I can also move the
create job method
from the routing file
to the controller

File Edit Selection View Go Run ...

workopia

EXPLORER

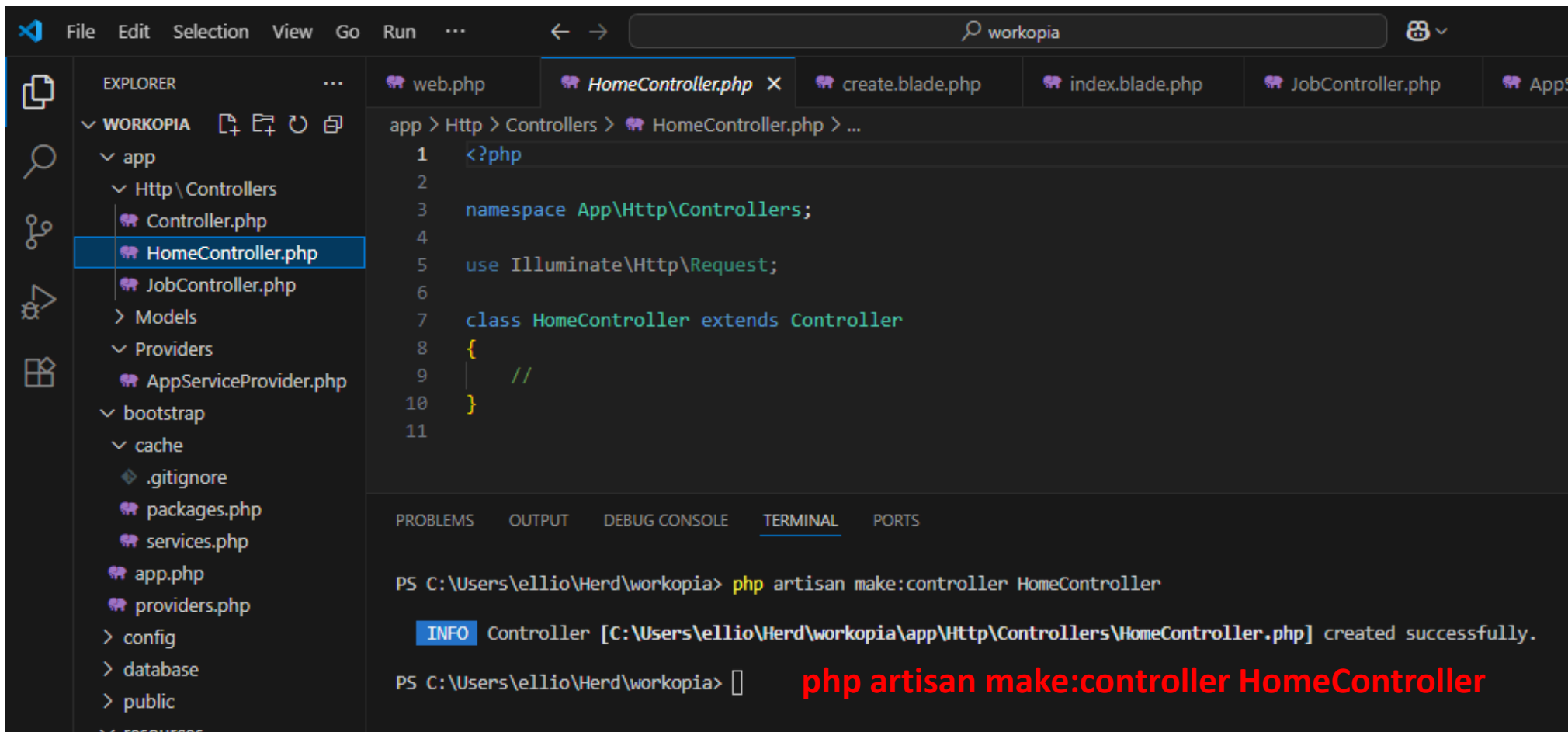
WORKOPIA

- resources
 - css
 - js
 - views
 - jobs
 - create.blade.php
 - index.blade.php
 - welcome.blade.php
 - routes
 - console.php

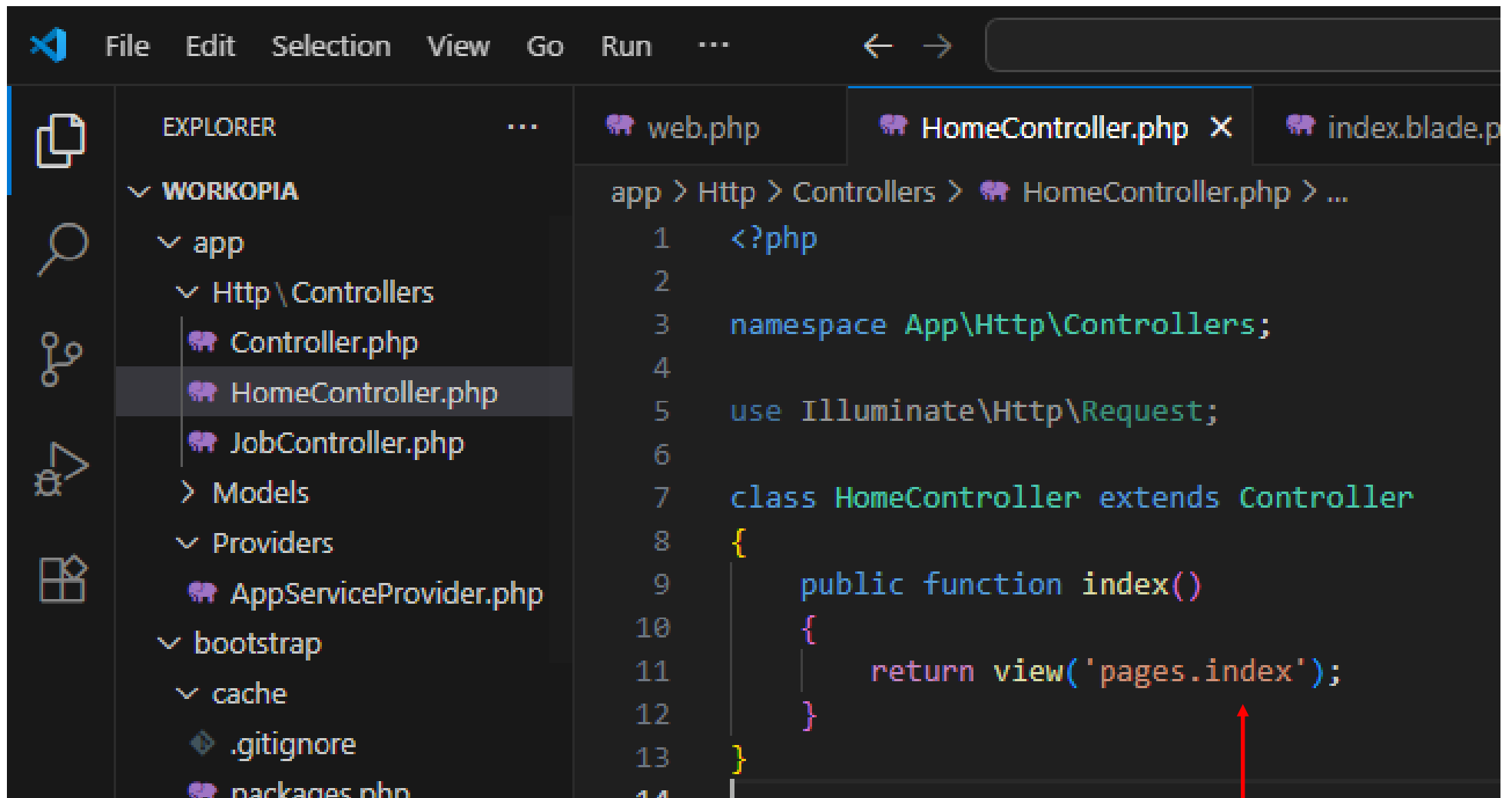
web.php

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\JobController;
5
6 Route::get('/', function () {
7     return view('welcome');
8 });
9
10 Route::get('jobs', [JobController::class, 'index']);
11 Route::get('jobs/create', [JobController::class, 'create']);
12
```

Then clean up the route so that the endpoint goes to the 'JobController' method

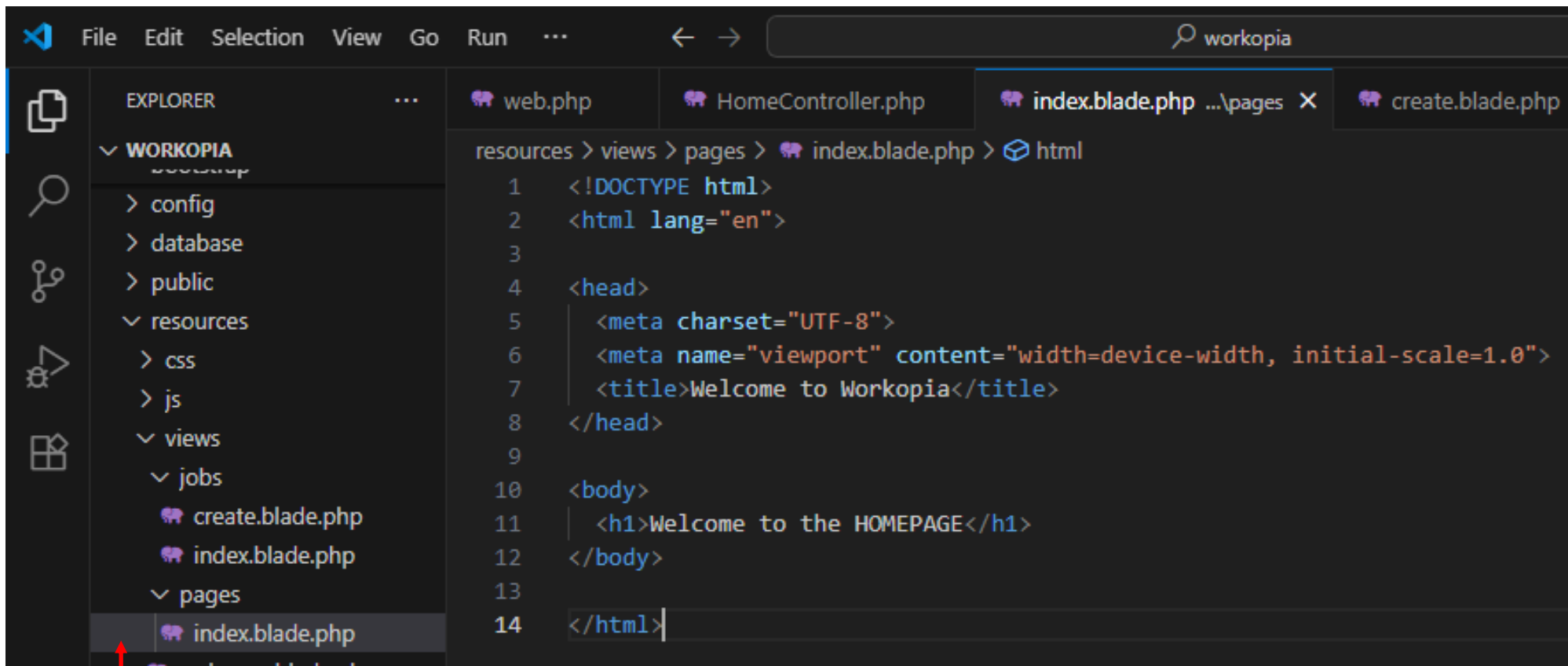


We could route the homepage to index.php but the homepage is going to have some special stuff on it like a hero gallery so it is worth making a 'HomeController'.



```
File Edit Selection View Go Run ...  
EXPLORER  
WORKOPIA  
app  
  Http \ Controllers  
    Controller.php  
    HomeController.php  
    JobController.php  
  Models  
  Providers  
    AppServiceProvider.php  
bootstrap  
  cache  
    .gitignore  
    packages.php  
web.php  
HomeController.php X  
index.blade.p  
app > Http > Controllers > HomeController.php > ...  
1 <?php  
2  
3 namespace App\Http\Controllers;  
4  
5 use Illuminate\Http\Request;  
6  
7 class HomeController extends Controller  
8 {  
9     public function index()  
10    {  
11        return view('pages.index');  
12    }  
13 }  
14
```

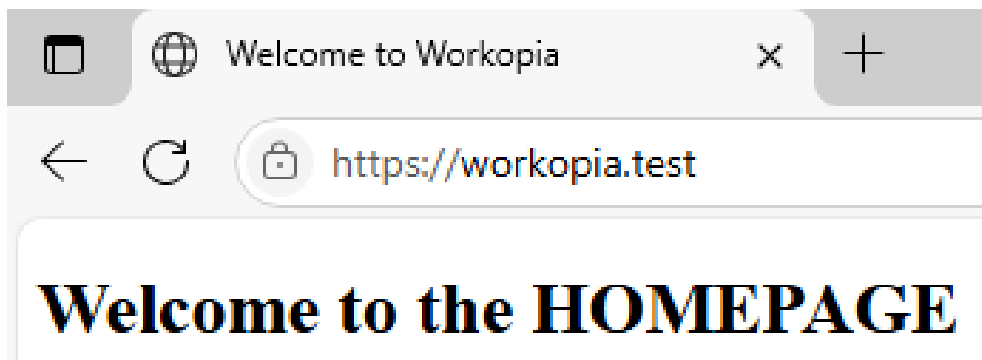
In the HomeController I creat an index method that points to a blade view called pages.index.



In the views folder, I have created a sub folder called pages and a file called 'index.blade.php' which is the html for the homepage.

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\JobController;
5 use App\Http\Controllers\HomeController;
6
7 Route::get('/', [HomeController::class, 'index']);
8 Route::get('jobs', [JobController::class, 'index']);
9 Route::get('jobs/create', [JobController::class, 'create']);
10
```

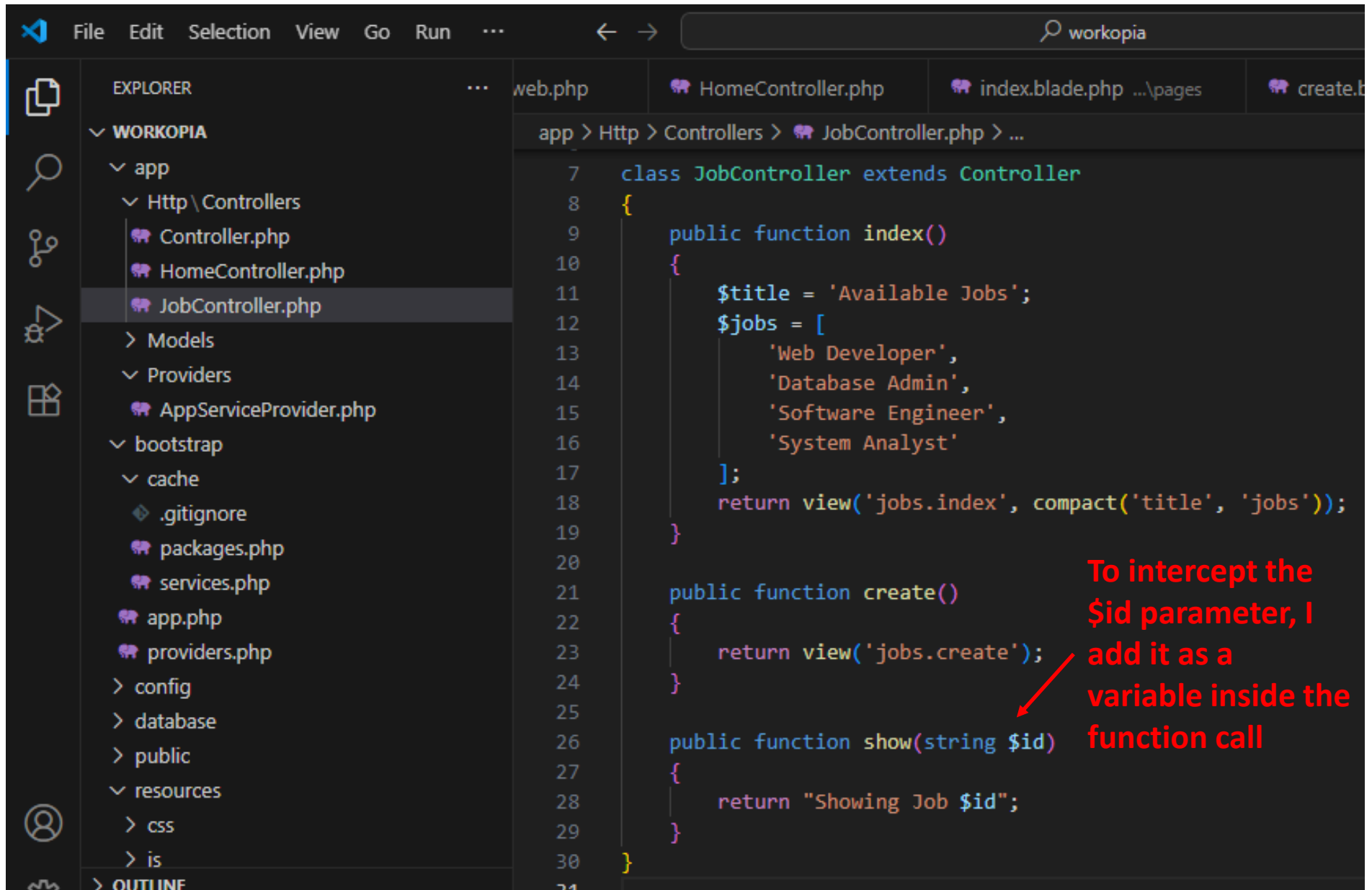
Add a route for the homepage '/' that points to the home controller index method



Now the homepage (or root) is landing on the correct view html.

Note how the web.php route file is now clean and only contains a list of pointers to the relevant controller and method. This is how a routing file should look.

3.6 Params & Requests in Controller



The screenshot shows an IDE with the Explorer panel on the left displaying the project structure. The file `JobController.php` is selected under `app > Http > Controllers`. The main editor shows the code for `JobController`, which extends `Controller`. The `index()` method returns a view with job titles. The `create()` method returns a view. The `show()` method takes a `$id` parameter and returns a string. A red arrow points to the `$id` parameter in the `show()` method signature, with a red text annotation: "To intercept the \$id parameter, I add it as a variable inside the function call".

```
7 class JobController extends Controller
8 {
9     public function index()
10    {
11        $title = 'Available Jobs';
12        $jobs = [
13            'Web Developer',
14            'Database Admin',
15            'Software Engineer',
16            'System Analyst'
17        ];
18        return view('jobs.index', compact('title', 'jobs'));
19    }
20
21    public function create()
22    {
23        return view('jobs.create');
24    }
25
26    public function show(string $id)
27    {
28        return "Showing Job $id";
29    }
30 }
```

To intercept the `$id` parameter, I add it as a variable inside the function call

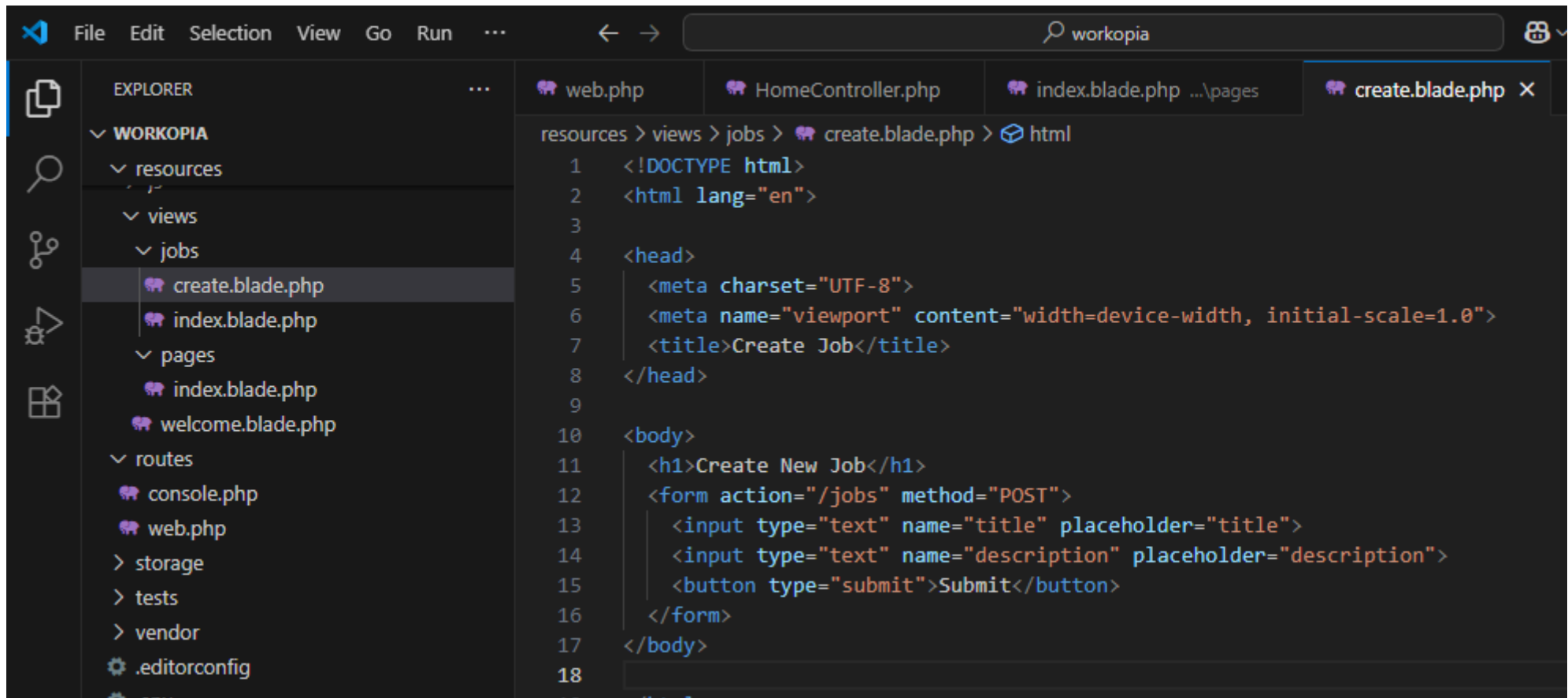
```
File Edit Selection View Go Run ... workopia

EXPLORER
WORKOPIA
  resources
  views
  jobs
    create.blade.php
    index.blade.php
  pages
    index.blade.php
    welcome.blade.php
  routes
    console.php
    web.php
  storage
  tests

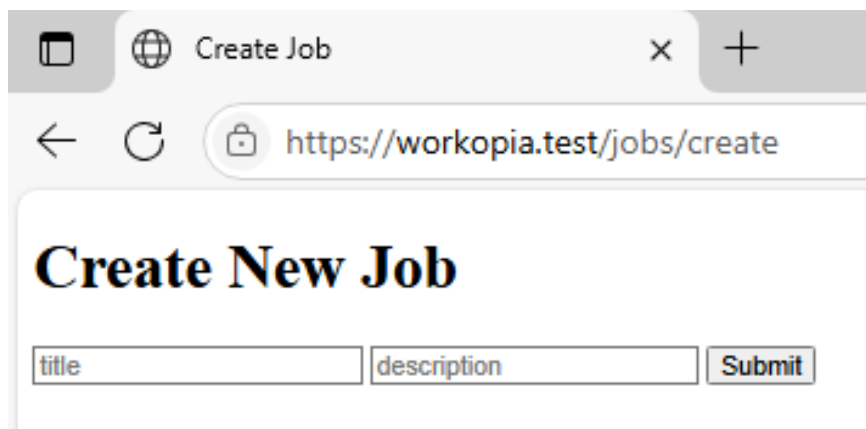
routes > web.php
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\JobController;
5 use App\Http\Controllers\HomeController;
6
7 Route::get('/', [HomeController::class, 'index']);
8 Route::get('jobs', [JobController::class, 'index']);
9 Route::get('jobs/create', [JobController::class, 'create']);
10 Route::get('jobs/{id}', [JobController::class, 'show']);
11
```

I now create a route from the endpoint. Note the id in curly braces to denote it is dynamic.

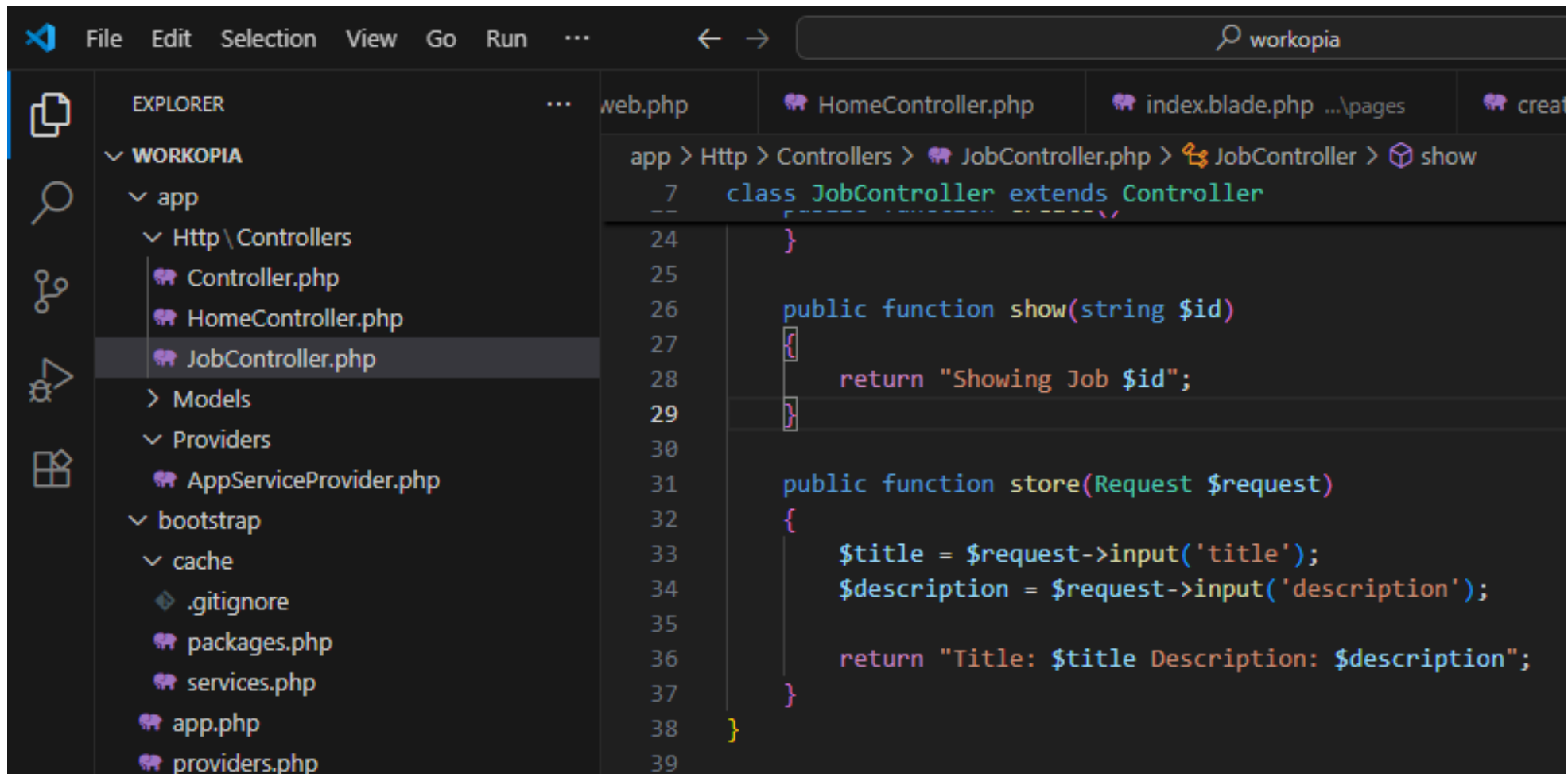
The ordering of routes in the route file is important. If I would place 'jobs/{id}' above '/jobs' then it would not route correctly to 'jobs' because there is a similar route that takes an additional dynamic parameter so it would go here. This is why the '/' root or homepage is at the top followed by Jobs, then jobs/create then jobs/{id}



```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5     <meta charset="UTF-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title>Create Job</title>
8 </head>
9
10 <body>
11     <h1>Create New Job</h1>
12     <form action="/jobs" method="POST">
13         <input type="text" name="title" placeholder="title">
14         <input type="text" name="description" placeholder="description">
15         <button type="submit">Submit</button>
16     </form>
17 </body>
18
```



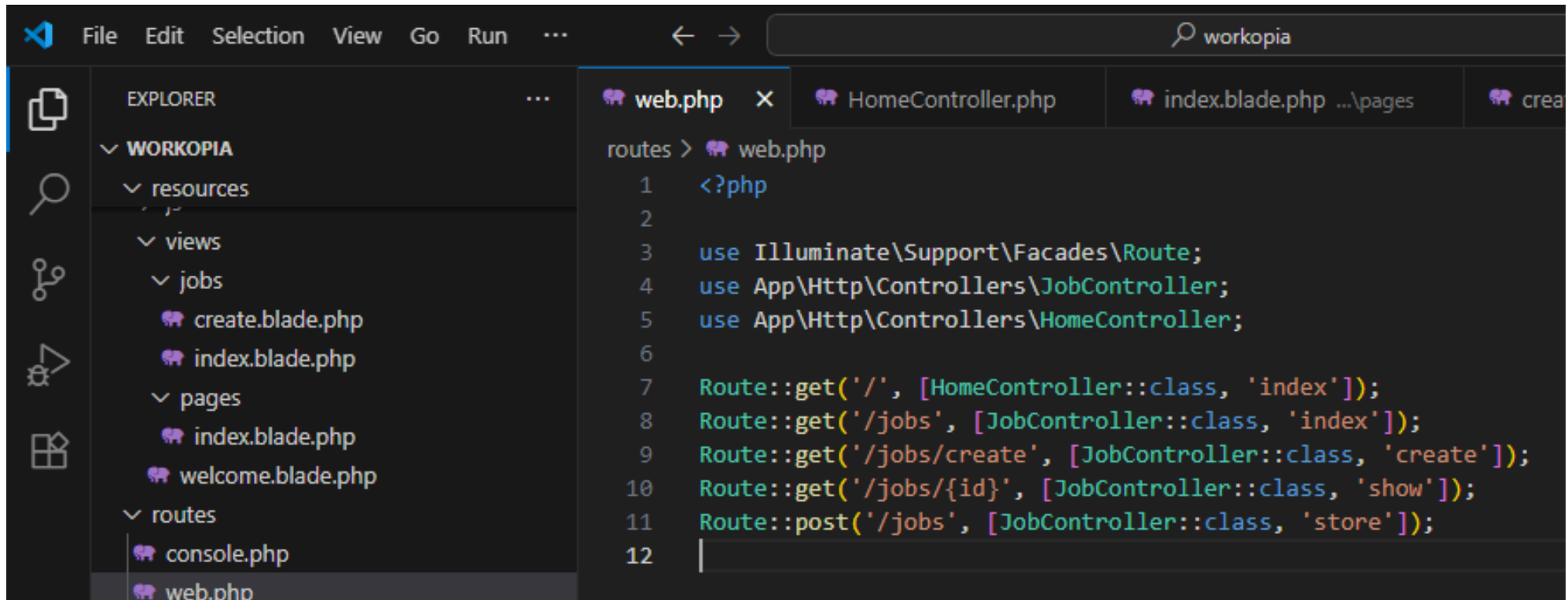
The 'jobs/create' view now has a simple form in it that passes a title and description value via POST to the '/jobs' page



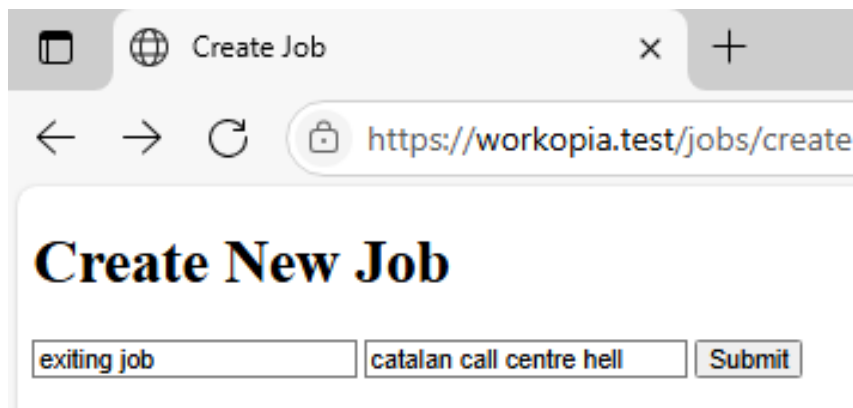
The screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left displaying the project structure. The file `JobController.php` is selected under `app/Http/Controllers`. The main editor area shows the code for `JobController.php`, which extends `Controller` and implements `show` and `store` methods.

```
7  class JobController extends Controller
24
25
26  public function show(string $id)
27  {
28      return "Showing Job $id";
29  }
30
31  public function store(Request $request)
32  {
33      $title = $request->input('title');
34      $description = $request->input('description');
35
36      return "Title: $title Description: $description";
37  }
38  }
39
```

In the 'JobController' I had a method called `store` to handle the form submission. Note that it takes in the request header info. In this simple example I am just pushing the POST params into variables and outputting them as a string.

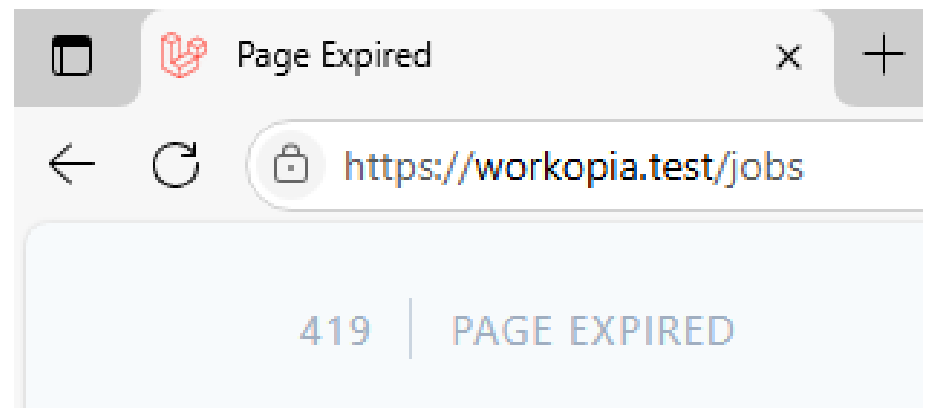


```
File Edit Selection View Go Run ... workopia
EXPLORER
WORKOPIA
  resources
  views
    jobs
      create.blade.php
      index.blade.php
    pages
      index.blade.php
      welcome.blade.php
  routes
    console.php
    web.php
routes > web.php
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\JobController;
5 use App\Http\Controllers\HomeController;
6
7 Route::get('/', [HomeController::class, 'index']);
8 Route::get('/jobs', [JobController::class, 'index']);
9 Route::get('/jobs/create', [JobController::class, 'create']);
10 Route::get('/jobs/{id}', [JobController::class, 'show']);
11 Route::post('/jobs', [JobController::class, 'store']);
12
```



But when I submit the form, it lands on a 419 page....

Now I add a route pointing to the controller and method.



workopia

EXPLORER

- WORKOPIA
 - resources
 - views
 - jobs
 - create.blade.php
 - index.blade.php
 - pages
 - index.blade.php
 - welcome.blade.php
 - routes
 - console.php
 - web.php
 - storage
 - app
 - framework
 - logs
 - tests
 - vendor

web.php HomeController.php index.blade.php ...\pages create.blade.php X

resources > views > jobs > create.blade.php > html

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Create Job</title>
8 </head>
9
10 <body>
11   <h1>Create New Job</h1>
12   <form action="/jobs" method="POST">
13     @csrf
14     <input type="text" name="title" placeholder="title">
15     <input type="text" name="description" placeholder="description">
16     <button type="submit">Submit</button>
17   </form>
18 </body>
19
20 </html>
```

It fails because Laravel uses CSRF protection so I need to use @csrf in the form.

title description

Submit

@csrf adds a hidden token into the form html. The token ensures that the data is actually from the form and not a malicious actor.

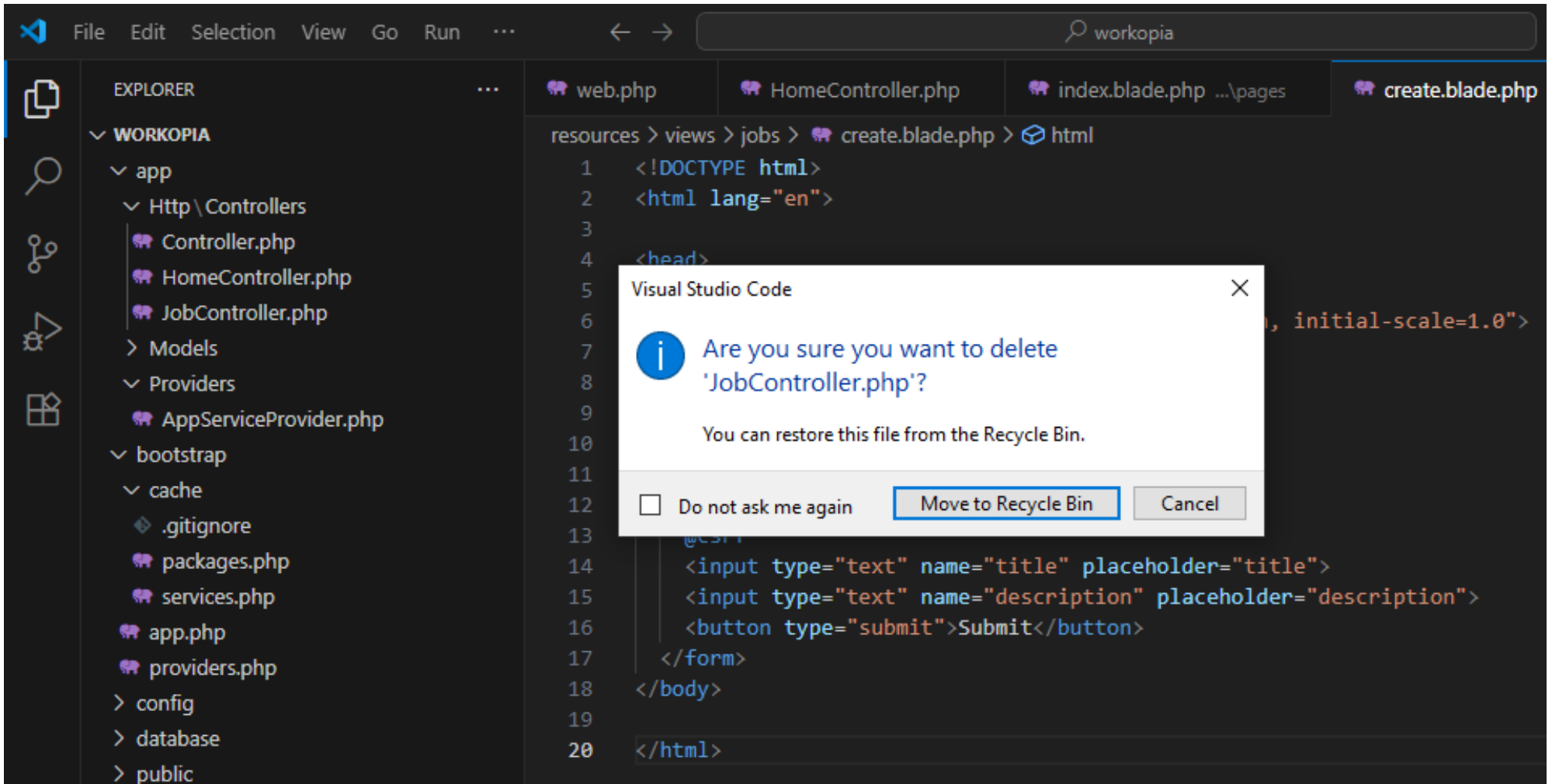
```
> <head> </head>
<body>
  <h1>Create New Job</h1>
  <form action="/jobs" method="POST">
    <input type="hidden" name="_token" value="g3q6oyzmmn5JbCKxYB5uTmaSBrnXavqEEdAbcOq" autocomplete="off">
    <input type="text" name="title" placeholder="title">
    <input type="text" name="description" placeholder="description">
    <button type="submit">Submit</button>
  </form>
</body>
</html>
```

3.7 Generate Resource Routes & Methods

Resource Methods

- | | |
|---|--|
| ✓ index() - List All Resources | ✓ edit() - Show Edit Form |
| ✓ show() - List Single Resource | ✓ PUT/PATCH
update() - Submit Edit Form |
| ✓ create() - Display Create Form | ✓ destroy() - Delete Resource |
| ✓ store() - Submit Create Form POST | DELETE |

The controller methods have a standard naming convention based on what they do.



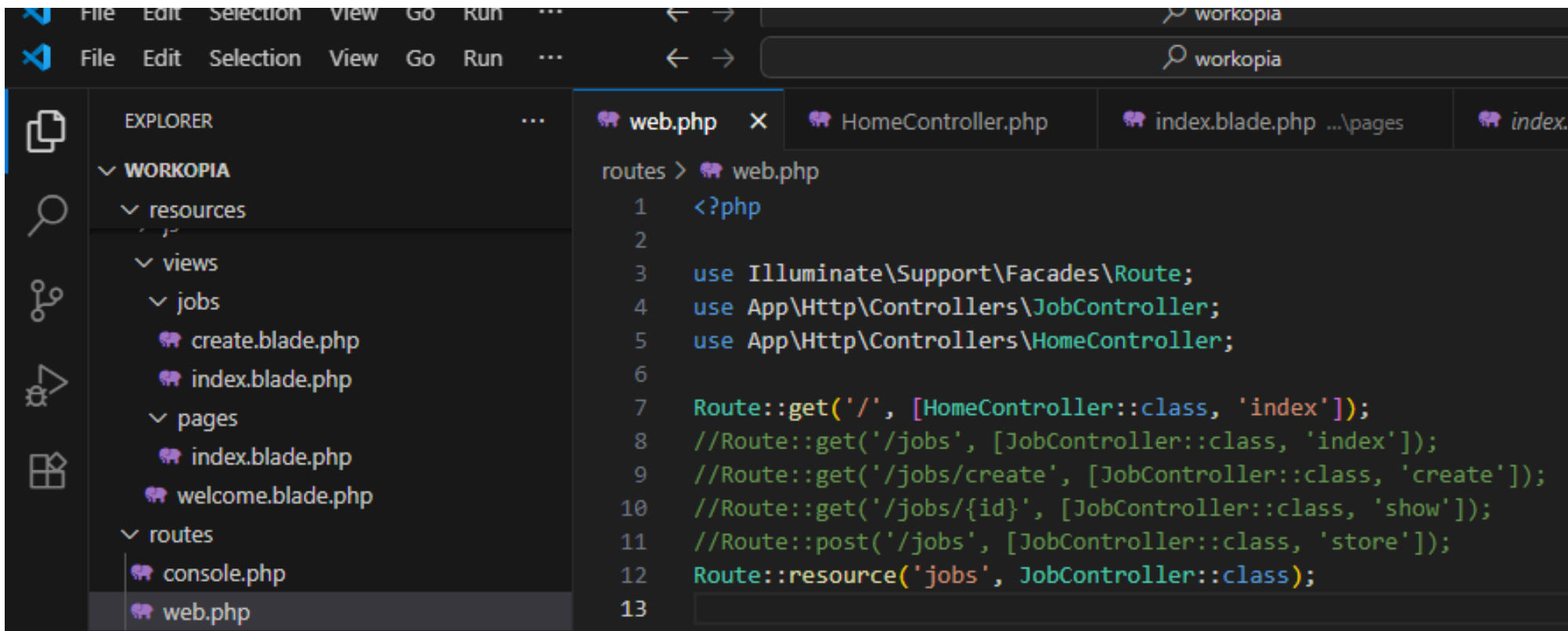
Because these controller methods are standardised, we can create them automatically using artisan so first I will delete the JobsController.

The screenshot displays an IDE interface with the following components:

- EXPLORER:** Shows the project structure. The 'app' directory is expanded, showing 'Http' > 'Controllers'. 'JobController.php' is selected.
- Code Editor:** Displays the content of 'JobController.php'. The code is as follows:

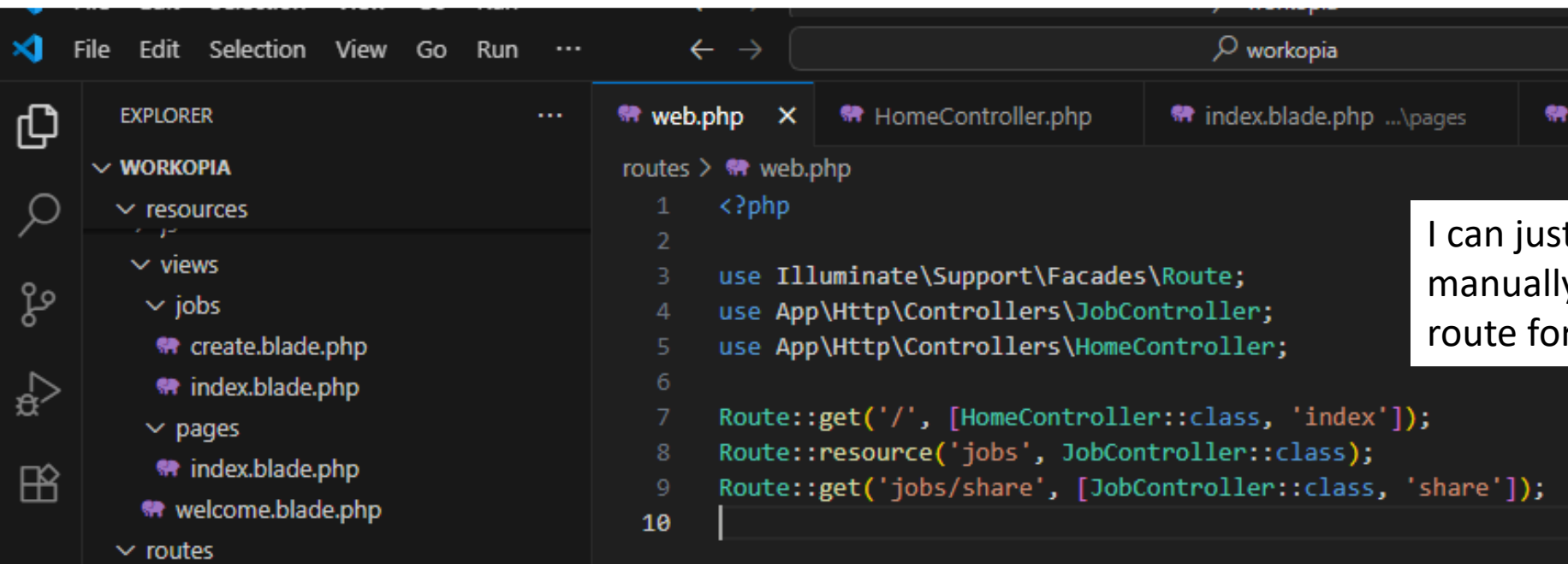
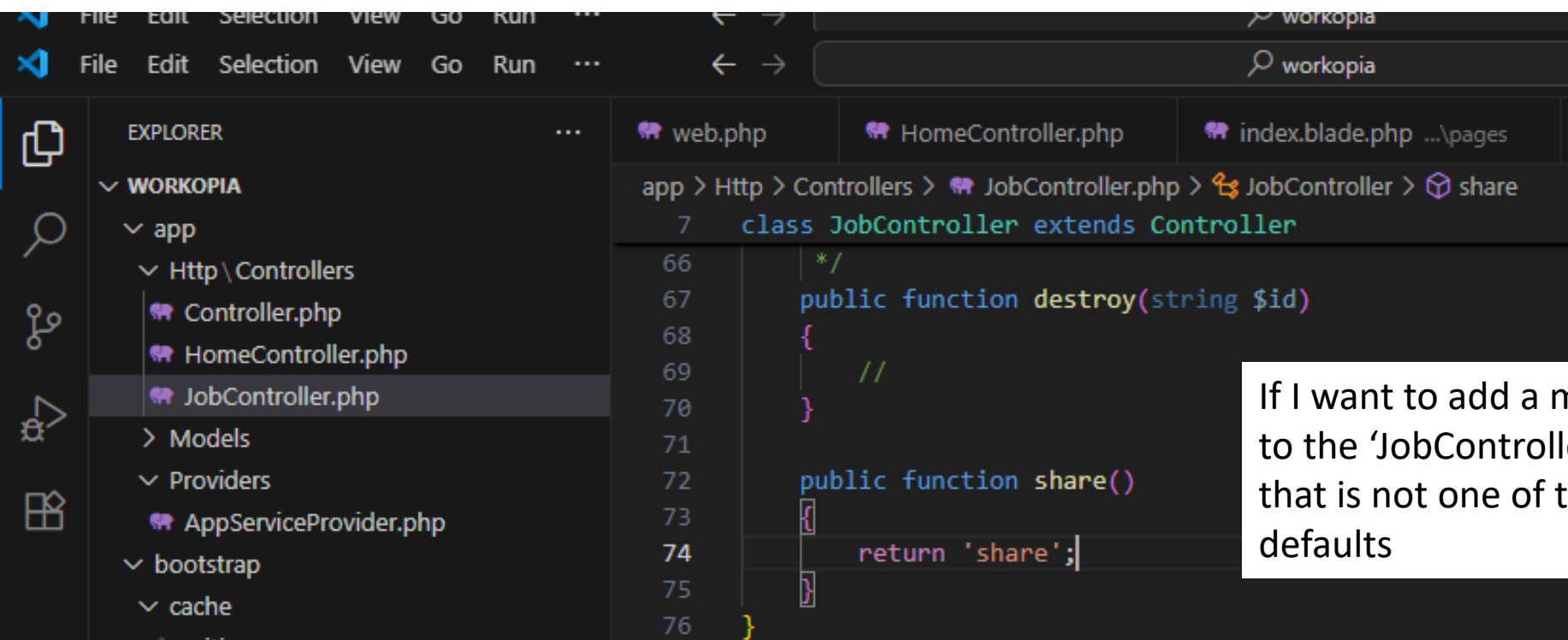
```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6
7 class JobController extends Controller
8 {
9     /**
10      * Display a listing of the resource.
11      */
12     public function index()
13     {
14         //
15     }
16 }
```
- TERMINAL:** Shows the command `php artisan make:controller JobController --resource` and the output: `INFO Controller [C:\Users\ellio\Herd\workopia\app\Http\Controllers\JobController.php] created successfully.`

Now when I create the controller via artisan in the terminal, it creates a template with all the default methods, ready for me to add the project specific code into each method.



```
File Edit Selection View Go Run ...  
File Edit Selection View Go Run ...  
workopia  
workopia  
EXPLORER  
WORKOPIA  
resources  
views  
jobs  
create.blade.php  
index.blade.php  
pages  
index.blade.php  
welcome.blade.php  
routes  
console.php  
web.php  
web.php X HomeController.php index.blade.php ...\pages index.  
routes > web.php  
1 <?php  
2  
3 use Illuminate\Support\Facades\Route;  
4 use App\Http\Controllers\JobController;  
5 use App\Http\Controllers\HomeController;  
6  
7 Route::get('/', [HomeController::class, 'index']);  
8 //Route::get('/jobs', [JobController::class, 'index']);  
9 //Route::get('/jobs/create', [JobController::class, 'create']);  
10 //Route::get('/jobs/{id}', [JobController::class, 'show']);  
11 //Route::post('/jobs', [JobController::class, 'store']);  
12 Route::resource('jobs', JobController::class);  
13
```

Now all the routing to the jobs controller can be removed and replaced with one line that routes to the resource of the 'JobsController' class.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell + v [ ] [ ] ... ^ X
PS C:\Users\ellio\Herd\workopia> php artisan route:list

GET|HEAD / ..... HomeController@index
GET|HEAD jobs ..... jobs.index > JobController@index
POST jobs ..... jobs.store > JobController@store
GET|HEAD jobs/create ..... jobs.create > JobController@create
GET|HEAD jobs/{job} ..... jobs.show > JobController@show
PUT|PATCH jobs/{job} ..... jobs.update > JobController@update
DELETE jobs/{job} ..... jobs.destroy > JobController@destroy
GET|HEAD jobs/{job}/edit ..... jobs.edit > JobController@edit
GET|HEAD storage/{path} ..... storage.local
GET|HEAD up .....

Showing [10] routes

PS C:\Users\ellio\Herd\workopia> 
```

I can see what routes have been created by `Route::resource('jobs', JobController::class);`
Using the vscode terminal 'php artisan route:list' command shows an expanded list of all the routes that come from this one line of code.

3.8 Type Hinting in Controllers

Common Types

- ✓ **Illuminate\Http\Request**
- ✓ **Illuminate\Http\Response**
- ✓ **Illuminate\Http\RedirectResponse**
- ✓ **Illuminate\Http\JsonResponse**
- ✓ **Illuminate\View\View**
- ✓ **Illuminate\Support\Collection**
- ✓ **Illuminate\Auth\Access\Response**

Type hinting is a way to specify the type of variable used in php and is a way to make tighter error free code.

It is not mandatory in Laravel but is good practice so going forward it will be used.

This is not the complete list but examples of the most common used ones.



File Edit Selection View Go Run ...



worko



EXPLORER



web.php

HomeController.php

index.blade.php ...\pages



WORKOPIA

app

Http\ Controllers

Controller.php

HomeController.php

JobController.php

Models

Providers

AppServiceProvider.php

bootstrap

cache

.gitignore

packages.php

services.php

app.php

providers.php

config

database

public

resources

app > Http > Controllers > JobController.php > ...

```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request; I will bring in the
6  use Illuminate\View\View; ← illuminate\View\View
7
8  class JobController extends Controller
9  {
10     public function index(): View
11     {
12         $jobs = [
13             'Web Developer',
14             'Database Admin',
15             'Software Engineer',
16             'System Analyst'
17         ];
18         Jobs.index will always return a
19         return view('jobs.index', compact('jobs'));
20     }
21
22     public function create()
```

File Edit Selection View Go Run ... workopia

EXPLORER

- WORKOPIA
 - app
 - Http \ Controllers
 - Controller.php
 - HomeController.php
 - JobController.php
 - Models
 - Providers
 - AppServiceProvider.php
 - bootstrap
 - cache
 - .gitignore
 - packages.php
 - services.php
 - app.php
 - providers.php
 - config
 - database
 - public
 - resources
 - css
 - js

web.php HomeController.php index.blade.php ... \pages index.bl

app > Http > Controllers > JobController.php > JobController

```
8 class JobController extends Controller
21
22 public function create(): View
23 {
24     return view('jobs.create');
25 }
26
27 public function store(Request $request): string
28 {
29     return 'store';
30 }
31
32 public function show(string $id): string
33 {
34     return 'show';
35 }
36
37 public function edit(string $id): string
38 {
39     return 'edit';
40 }
41
42 public function update(Request $request, string $id): string
43 {
```

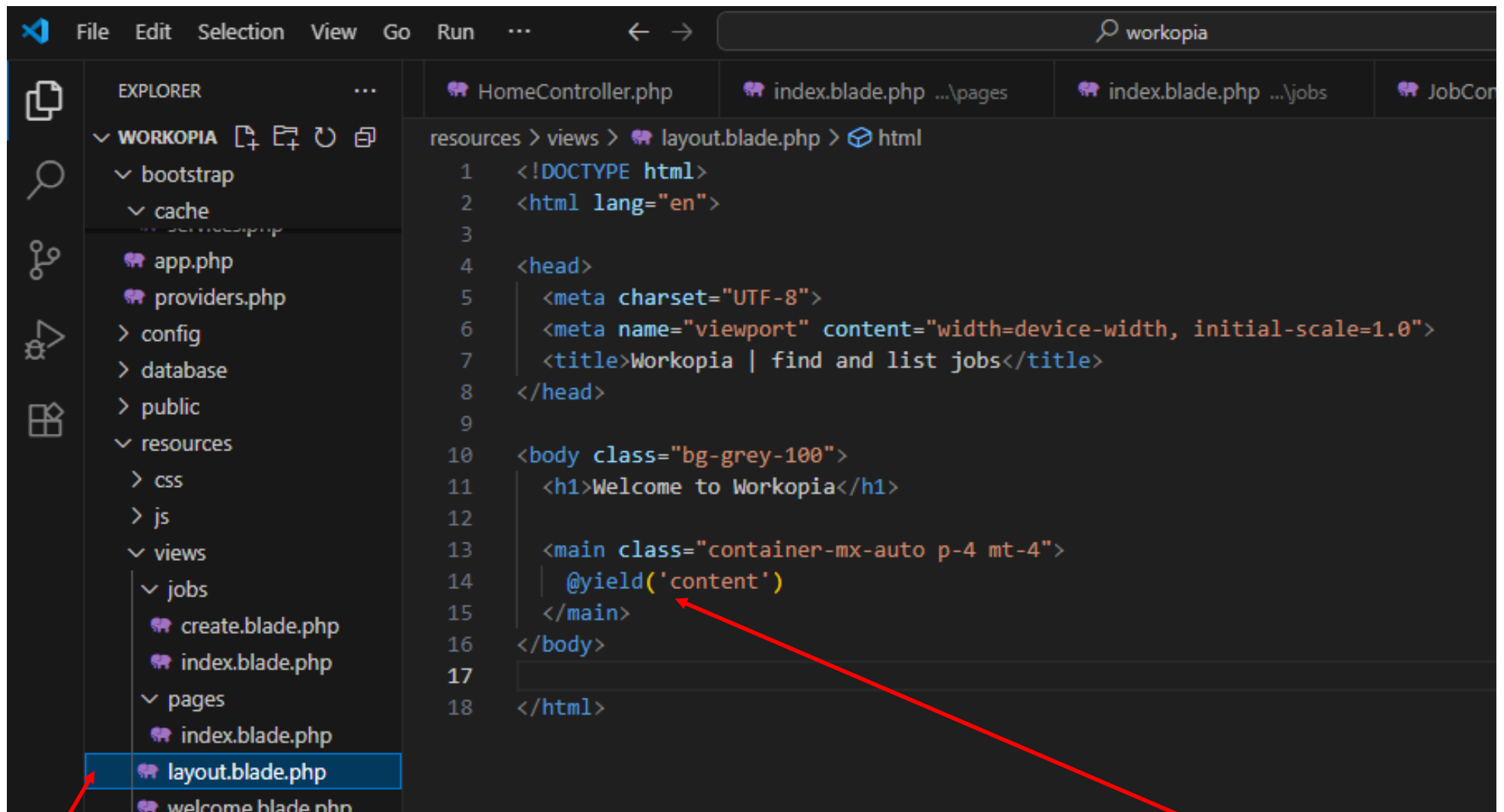
Create is returning a View

The rest are temporarily set to return a string

3.9 Layouts with Template Inheritance

The views all have the same or very similar html markup code so templates can be used so that I don't have to repeat myself typing the same code over again and again. Templates is more of an old-fashioned way of doing things with layout components and slots being the more UX focused way of doing things.

This project will be written using the more modern layouts and slots method but is worth understanding templates because there are websites written in Laravel that still use this method.



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a project structure for 'WORKOPIA'. The 'views' folder is expanded, showing subfolders 'jobs' and 'pages', and files 'create.blade.php', 'index.blade.php', and 'layout.blade.php'. The 'layout.blade.php' file is selected and highlighted in blue. A red arrow points from the text below to this file. The main editor area shows the content of 'layout.blade.php', which is an HTML boilerplate. It includes a DOCTYPE declaration, a lang attribute, a head section with charset and viewport meta tags, a title 'Workopia | find and list jobs', and a body section with a class 'bg-grey-100'. Inside the body, there is an h1 'Welcome to Workopia' and a main container with the class 'container-mx-auto p-4 mt-4'. The container contains the '@yield('content')' directive. A red arrow points from the text below to this directive. The breadcrumb at the top of the editor shows the path: 'resources > views > layout.blade.php > html'.

```
resources > views > layout.blade.php > html
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Workopia | find and list jobs</title>
8  </head>
9
10 <body class="bg-grey-100">
11     <h1>Welcome to Workopia</h1>
12
13     <main class="container-mx-auto p-4 mt-4">
14         @yield('content')
15     </main>
16 </body>
17
18 </html>
```

First I create a layout.blade.php file at the top level of the view folder. I then paste in my boilerplate html markup and add some classes to the body and main tags to later use with tailwinds.css. Within the main container is where I want the page content (view), so I have to use the @yield directive.

The screenshot shows the VS Code interface. On the left, the Explorer sidebar displays a project structure with folders 'resources', 'views', 'jobs', 'pages', and 'routes'. The file 'index.blade.php' is selected under the 'jobs' folder. The main editor area shows the content of 'index.blade.php' with line numbers 1 through 13. The code is as follows:

```
1 @extends('layout')
2
3 @section('content')
4     <h2>Available Jobs</h2>
5     <ul>
6         @forelse($jobs as $job)
7             <li>{{ $job }}</li>
8         @empty
9             <li>No Jobs Available</li>
10        @endforelse
11    </ul>
12 @endsection
13
```

To use the layout in the views I have to extend it

I also have to specify where I want to 'yield' the content to using the section directive

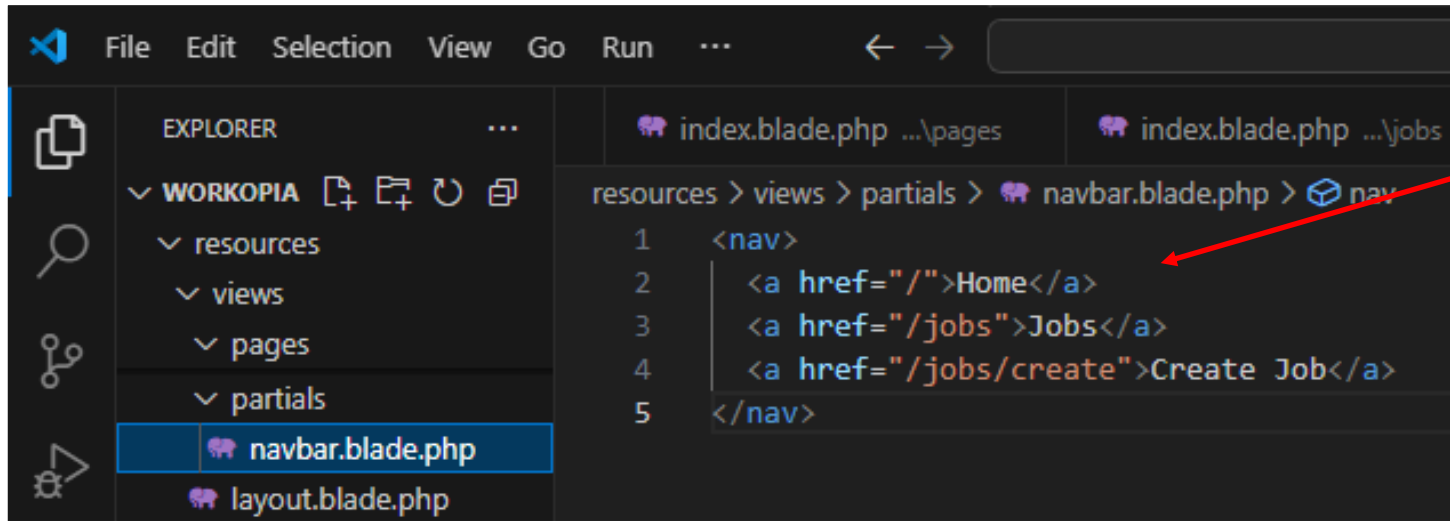
It is important to close out the section directive.

This screenshot is similar to the first one, showing the same VS Code interface and file structure. However, the code in the editor is slightly different, with the section directive properly closed at line 12:

```
1 @extends('layout')
2
3 @section('content')
4     <h2>Available Jobs</h2>
5     <ul>
6         @forelse($jobs as $job)
7             <li>{{ $job }}</li>
8         @empty
9             <li>No Jobs Available</li>
10        @endforelse
11    </ul>
12 @endsection
```


3.10 Partials & @include directive

A partial is like a mini block of html that I want on each page or want to use more than once, for example a navbar. I create a 'partials' folder under views where I can put all the partial files



The screenshot shows the VS Code interface. On the left, the Explorer sidebar shows a project structure with folders 'resources', 'views', 'pages', and 'partials'. The 'partials' folder is expanded, showing 'navbar.blade.php' and 'layout.blade.php'. The main editor window shows the content of 'navbar.blade.php' with the following code:

```
1 <nav>
2   <a href="/">Home</a>
3   <a href="/jobs">Jobs</a>
4   <a href="/jobs/create">Create Job</a>
5 </nav>
```

A red arrow points from the text 'I create a 'partials' folder under views where I can put all the partial files' to the 'partials' folder in the Explorer.



The screenshot shows the VS Code interface. On the left, the Explorer sidebar shows the same project structure, but now 'layout.blade.php' is selected. The main editor window shows the content of 'layout.blade.php' with the following code:

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>@yield('title', 'Workopia | find and list jobs')</title>
8 </head>
9
10 <body class="bg-grey-100">
11   @include('partials.navbar')
12
13   <main class="container-mx-auto p-4 mt-4">
14     @yield('content')
15   </main>
16 </body>
17
18 </html>
```

A red arrow points from the text 'I @include the partial in the layout.blade.php file at the point where I want it to appear.' to the '@include('partials.navbar')' line in the code.

I @include the partial in the layout.blade.php file at the point where I want it to appear.

4. Components & Styling

[4.1 What are Components?](#)

[4.2 Layout Component & Slots](#)

[4.3 Tailwind CSS & Vite Hot Reloading](#)

[4.4 Header Component & URL Helper](#)

[4.5 Conditional Classes & @php directive](#)

[4.6 Component attributes & Props](#)

[4.7 Button Link Component Challenge](#)

[4.8 Mobile Menu Nav Link](#)

[4.9 Mobile Menu Toggle](#)

[4.10 Hero Component](#)

[4.11 Top & Bottom Banners](#)

4.1 What are Components?

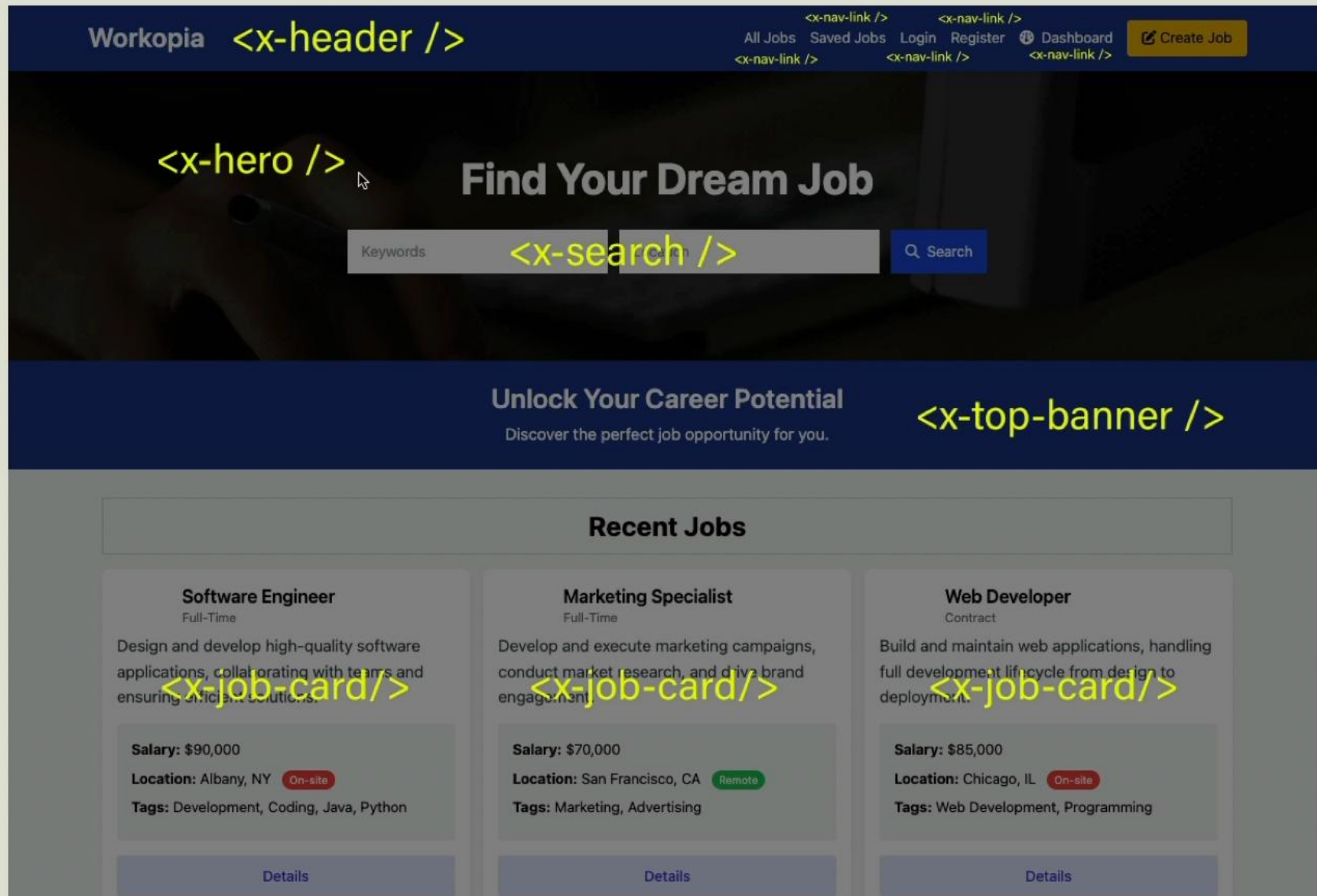


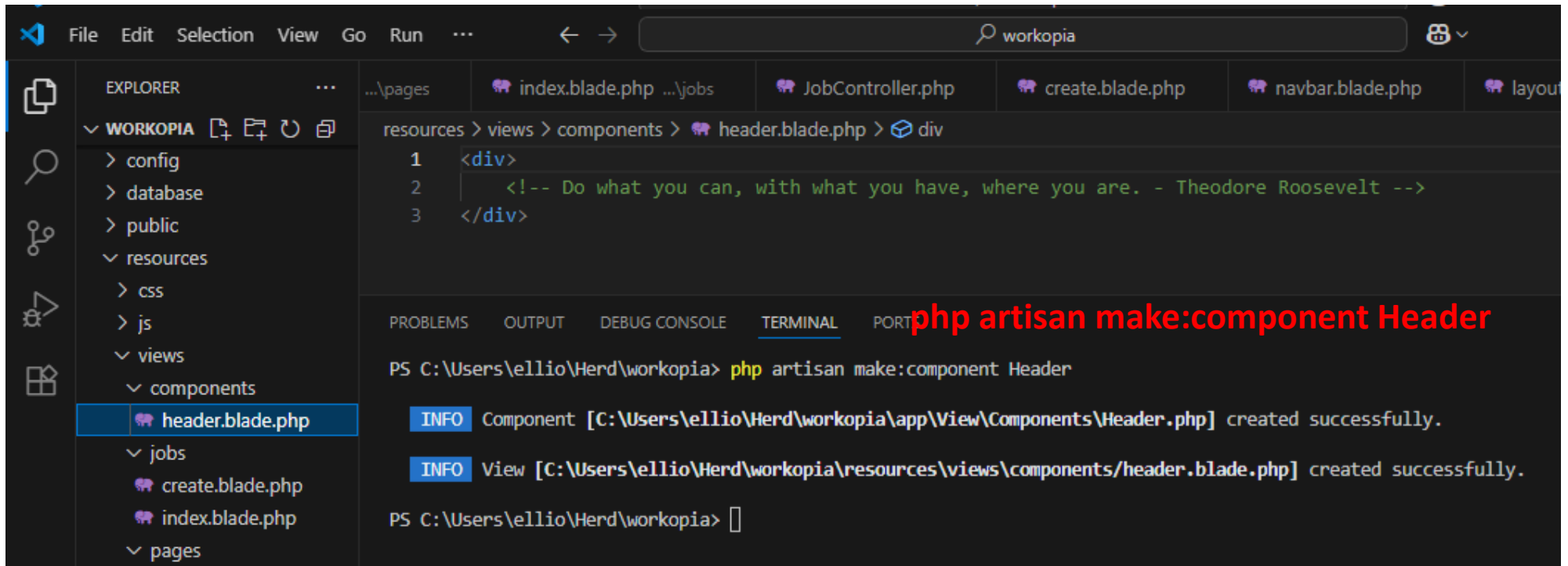
Laravel From Scratch

Components & Styling - Section Intro

- ✓ **Intro To Components**
- ✓ **Component Layouts**
- ✓ **Slots & Named Slots**
- ✓ **Tailwind CSS Setup**
- ✓ **Header Component**
- ✓ **Attributes & Props**
- ✓ **Public Assets**
- ✓ **Nav Link Component**
- ✓ **@php Helper & More**

UI Components





I can create components manually but there is a php artisan command that creates the file and folder structure.

File Edit Selection View Go Run ... workopia

EXPLORER

- WORKOPIA
 - config
 - database
 - public
 - resources
 - css
 - js
 - views
 - components
 - header.blade.php
 - jobs
 - create.blade.php
 - index.blade.php
 - pages
 - index.blade.php
 - layout.blade.php
 - welcome.blade.php

resources > views > layout.blade.php > html > body.bg-grey-100 > x-header

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>@yield('title', 'Workopia | find and list jobs')</title>
8 </head>
9
10 <body class="bg-grey-100">
11   <x-header/>
12
13   <main class="container-mx-auto p-4 mt-4">
14     @yield('content')
15   </main>
16 </body>
17
18 </html>
```

Now I can remove the @include navbar and replace it with an x-header to use the new header component

File Edit Selection View Go Run ...

workopia

WORKOPIA

app

Http \ Controllers

Controller.php

HomeController.php

JobController.php

Models

Providers

AppServiceProvider.php

View \ Components

Header.php

bootstrap

cache

.gitignore

packages.php

services.php

app.php

providers.php

config

database

public

resources

OUTLINE

index.blade.php ...\pages

index.blade.php ...\jobs

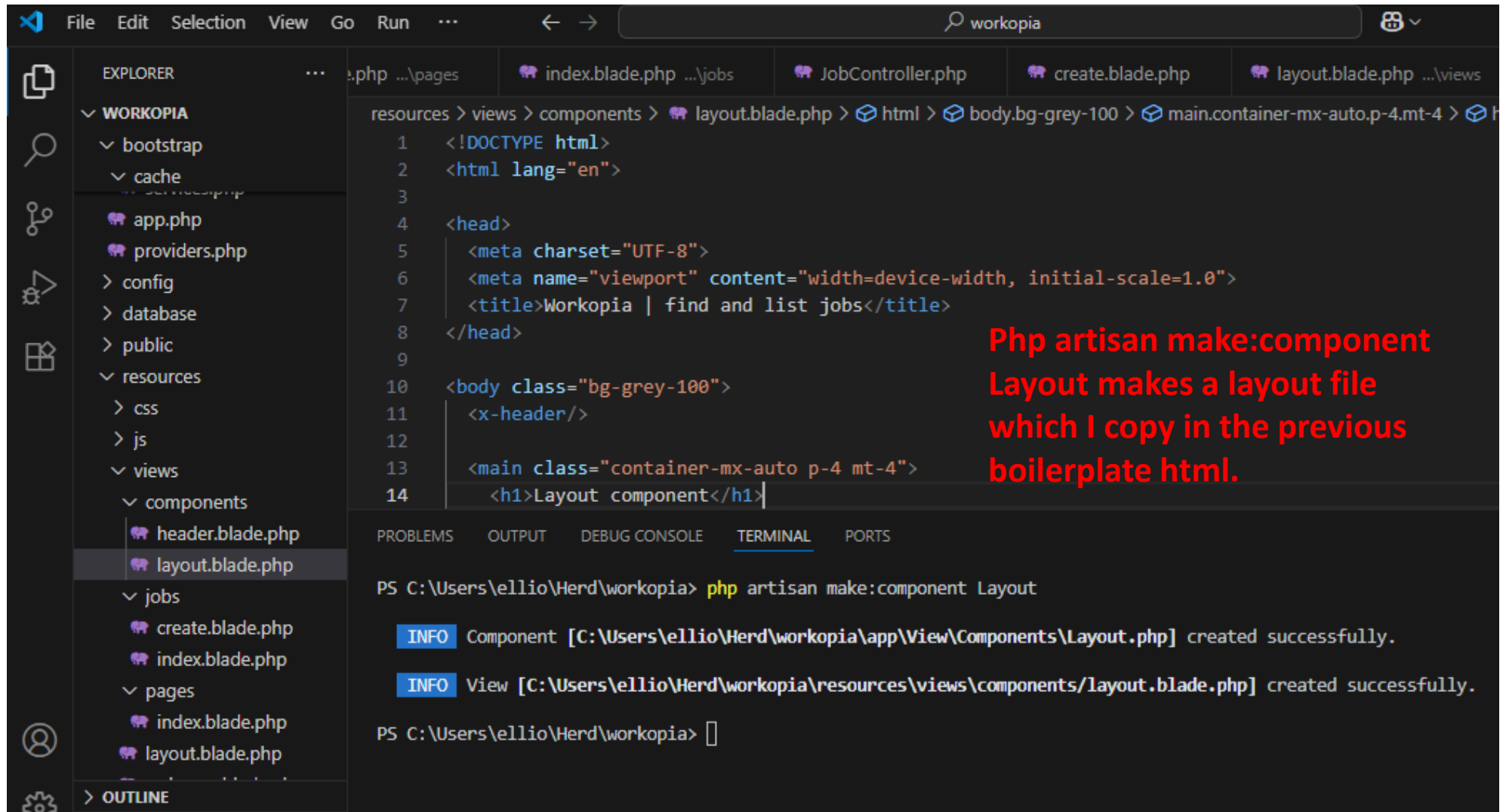
JobController.php

app > View > Components > Header.php > Header > __construct

```
1 <?php
2
3 namespace App\View\Components;
4
5 use Closure;
6 use Illuminate\Contracts\View\View;
7 use Illuminate\View\Component;
8
9 class Header extends Component
10 {
11     /**
12      * Create a new component instance.
13      */
14     public function __construct()
15     {
16         //
17     }
18
19     /**
20      * Get the view / contents that represent the component.
21      */
22     public function render(): View|Closure|string
23     {
24         return view('components.header');
25     }
26 }
```

Note that in the App/view/components / folder a header.php class is created that we can use to pass in data or initialisation

4.2 Layout Components & Slots



The screenshot shows the Visual Studio Code interface with a project named 'workopia'. The Explorer sidebar on the left shows the file structure, with the 'components' folder under 'views' selected. The main editor displays the content of 'layout.blade.php' in the path 'resources > views > components > layout.blade.php'. The code is as follows:

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Workopia | find and list jobs</title>
8 </head>
9
10 <body class="bg-grey-100">
11   <x-header/>
12
13   <main class="container-mx-auto p-4 mt-4">
14     <h1>Layout component</h1>
```

Below the code editor, the TERMINAL panel shows the execution of the command `php artisan make:component Layout`. The output indicates that the component and view were created successfully:

```
PS C:\Users\ellio\Herd\workopia> php artisan make:component Layout

[INFO] Component [C:\Users\ellio\Herd\workopia\app\View\Components\Layout.php] created successfully.

[INFO] View [C:\Users\ellio\Herd\workopia\resources\views\components\layout.blade.php] created successfully.

PS C:\Users\ellio\Herd\workopia>
```

Php artisan make:component Layout makes a layout file which I copy in the previous boilerplate html.

File Edit Selection View Go Run ...

workopia

WORKOPIA

> config

> database

> public

> resources

> css

> js

> views

> components

header.blade.php

layout.blade.php

> jobs

create.blade.php

index.blade.php

> pages

index.blade.php

welcome.blade.php

> routes

index.blade.php ...\pages

index.blade.php ...\jobs

JobController.php

cr

resources > views > components > layout.blade.php > html > body.bg-grey-100 > main.cont

1 <!DOCTYPE html>

2 <html lang="en">

3

4 <head>

5 <meta charset="UTF-8">

6 <meta name="viewport" content="width=device-width, initial-scale=1.0">

7 <title>Workopia | find and list jobs</title>

8 </head>

9

10 <body class="bg-grey-100">

11 <x-header/>

12

13 <main class="container-mx-auto p-4 mt-4">

14 <h1>Layout component</h1>

15 {{ \$slot }}

16 </main>

17 </body>

18

19 </html>

20

I use {{ \$slot }}

to denote where I

want the dynamic content to be

placed in the layout template.

The screenshot shows the VS Code interface with the Explorer on the left and the Editor on the right. The Explorer shows a project named 'WORKOPIA' with a file structure: resources > views > jobs > index.blade.php. The Editor shows the content of 'index.blade.php' in the 'jobs' view, which extends the 'x-layout' layout. The code is as follows:

```
1 <x-layout>
2   <h2>Available Jobs</h2>
3   <ul>
4     @foreach($jobs as $job)
5       <li>{{ $job }}</li>
6     @empty
7       <li>No Jobs Available</li>
8     @endforeach
9   </ul>
10 </x-layout>
```

Red arrows point from the text 'I wrap the dynamic content for each view in an <x-layout> opening and closing tag.' to the opening and closing <x-layout> tags.

I wrap the dynamic content for each view in an <x-layout> opening and closing tag.

The screenshot shows the VS Code interface with the Explorer on the left and the Editor on the right. The Explorer shows a project named 'WORKOPIA' with a file structure: resources > views > jobs > create.blade.php. The Editor shows the content of 'create.blade.php' in the 'jobs' view, which uses the 'x-layout' layout. The code is as follows:

```
1 <x-layout>
2   @section('content')
3     <h2>Create New Job</h2>
4     <form action="/jobs" method="POST">
5       @csrf
6       <input type="text" name="title" placeholder="title">
7       <input type="text" name="description" placeholder="description">
8       <button type="submit">Submit</button>
9     </form>
10 </x-layout>
```

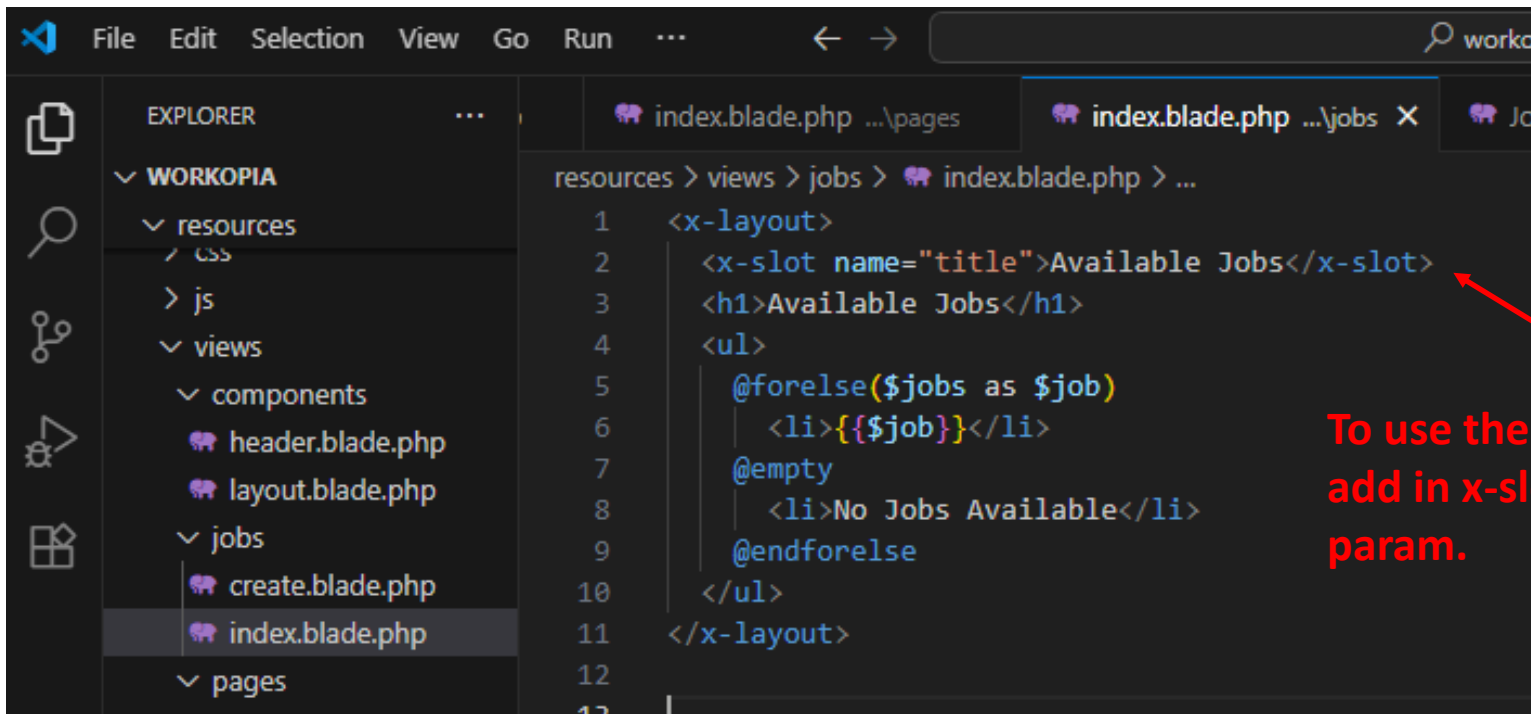
Red arrows point from the text 'This is a lot cleaner than @extends' to the opening and closing <x-layout> tags.

This is a lot cleaner than @extends

The screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left. The file explorer is expanded to show the 'views' directory, specifically the 'components' subdirectory, where 'layout.blade.php' is selected. The main editor area displays the content of 'layout.blade.php', which is a Blade template. The template includes a DOCTYPE declaration, an HTML lang attribute, a head section with meta tags for charset and viewport, and a title tag. The title tag uses a named slot: `<title>{{$title ?? 'Workopia | find and list jobs'}}</title>`. A red arrow points from a text box to this line of code. The text box contains the following explanation:

To add a custom title, I can use a NAMED SLOT. The double question marks denotes that if no title variable is found, use the default.

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>{{$title ?? 'Workopia | find and list jobs'}}</title>
8 </head>
9
10 <body class="bg-grey-100">
11   <x-header/>
12
13   <main class="container-mx-auto p-4 mt-4">
14     {{$slot}}
15   </main>
16 </body>
17
18 </html>
```

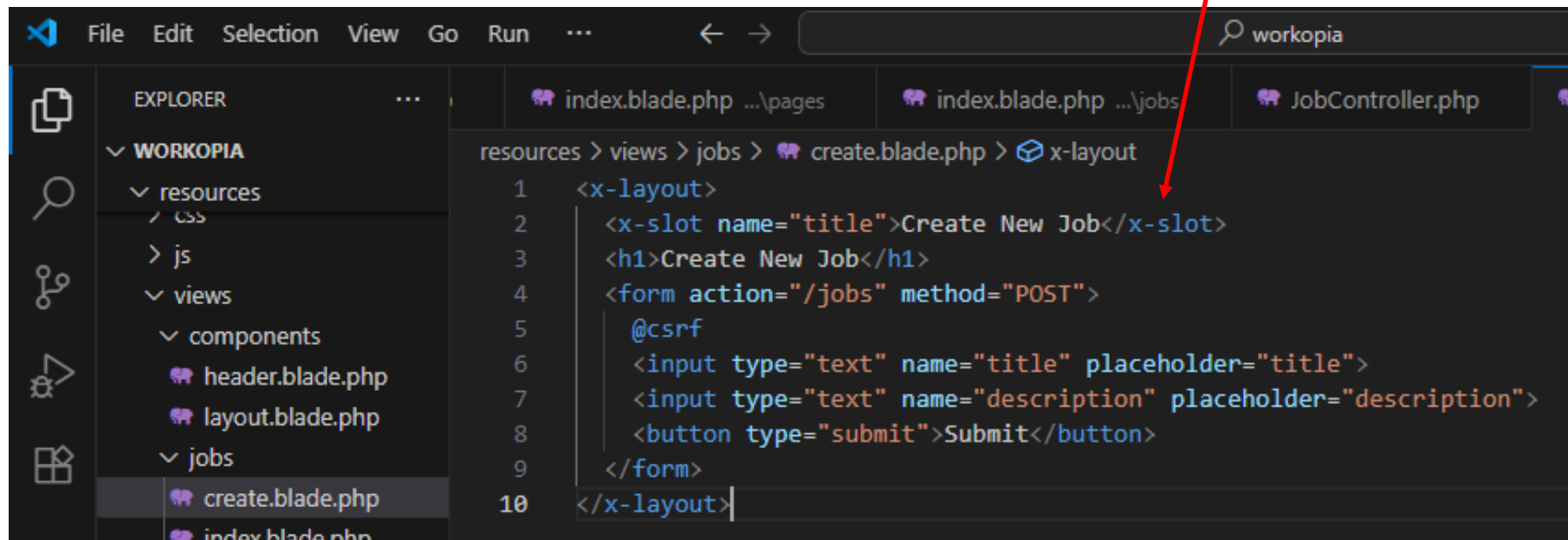


The screenshot shows the Visual Studio Code editor with the Explorer sidebar on the left. The Explorer is expanded to show the 'resources > views > jobs' directory. The file 'index.blade.php' is selected and open in the editor. The code in the editor is as follows:

```
1 <x-layout>
2   <x-slot name="title">Available Jobs</x-slot>
3   <h1>Available Jobs</h1>
4   <ul>
5     @foreach($jobs as $job)
6       <li>{{$job}}</li>
7     @empty
8       <li>No Jobs Available</li>
9     @endforeach
10  </ul>
11 </x-layout>
```

A red arrow points from the text 'To use the NAMED SLOT, I just add in x-slot tags with a name param.' to the `<x-slot name="title">` tag in the code.

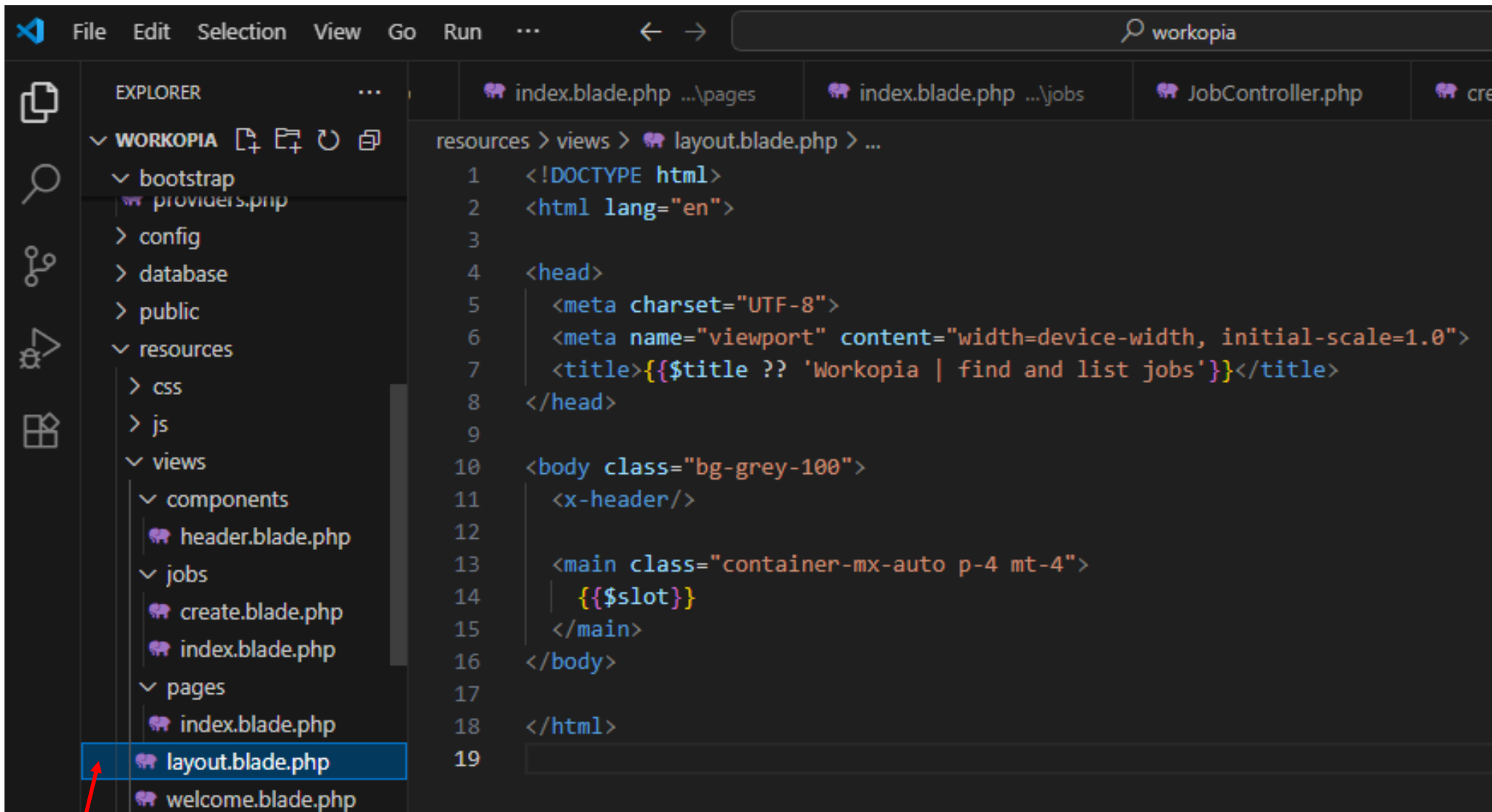
To use the NAMED SLOT, I just add in x-slot tags with a name param.



The screenshot shows the Visual Studio Code editor with the Explorer sidebar on the left. The Explorer is expanded to show the 'resources > views > jobs' directory. The file 'create.blade.php' is selected and open in the editor. The code in the editor is as follows:

```
1 <x-layout>
2   <x-slot name="title">Create New Job</x-slot>
3   <h1>Create New Job</h1>
4   <form action="/jobs" method="POST">
5     @csrf
6     <input type="text" name="title" placeholder="title">
7     <input type="text" name="description" placeholder="description">
8     <button type="submit">Submit</button>
9   </form>
10 </x-layout>
```

A red arrow points from the text 'To use the NAMED SLOT, I just add in x-slot tags with a name param.' to the `<x-slot name="title">` tag in the code.



For personal choice, the layout.blade.php is moved to the root of the views folder. Any UI components will be in the components folder and any page specific layout will be in a separate folder for the page.

File

Edit

Selection

View

Go

Run

...

←

→

workopia

WORKOPIA

app

Http\

Controllers

Controller.php

HomeController.php

JobController.php

Models

Providers

AppServiceProvider.php

View\

Components

Header.php

Layout.php

bootstrap

cache

.gitignore

packages.php

services.php

app.php

providers.php

config

database

public

OUTLINE

TIMELINE

index.blade.php ...\pages

index.blade.php ...\jobs

JobController.php

app > View > Components > Layout.php > Layout > render

1

<?php

2

3

namespace App\View\Components;

4

5

use Closure;

6

use Illuminate\Contracts\View\View;

7

use Illuminate\View\Component;

8

9

class Layout extends Component

10

{

11

/**

12

* Create a new component instance.

13

*/

14

public function __construct()

15

{

16

//

17

}

18

19

/**

20

* Get the view / contents that represent the component.

21

*/

22

public function render(): View|Closure|string

23

{

24

return view('layout');

25

}

26

}

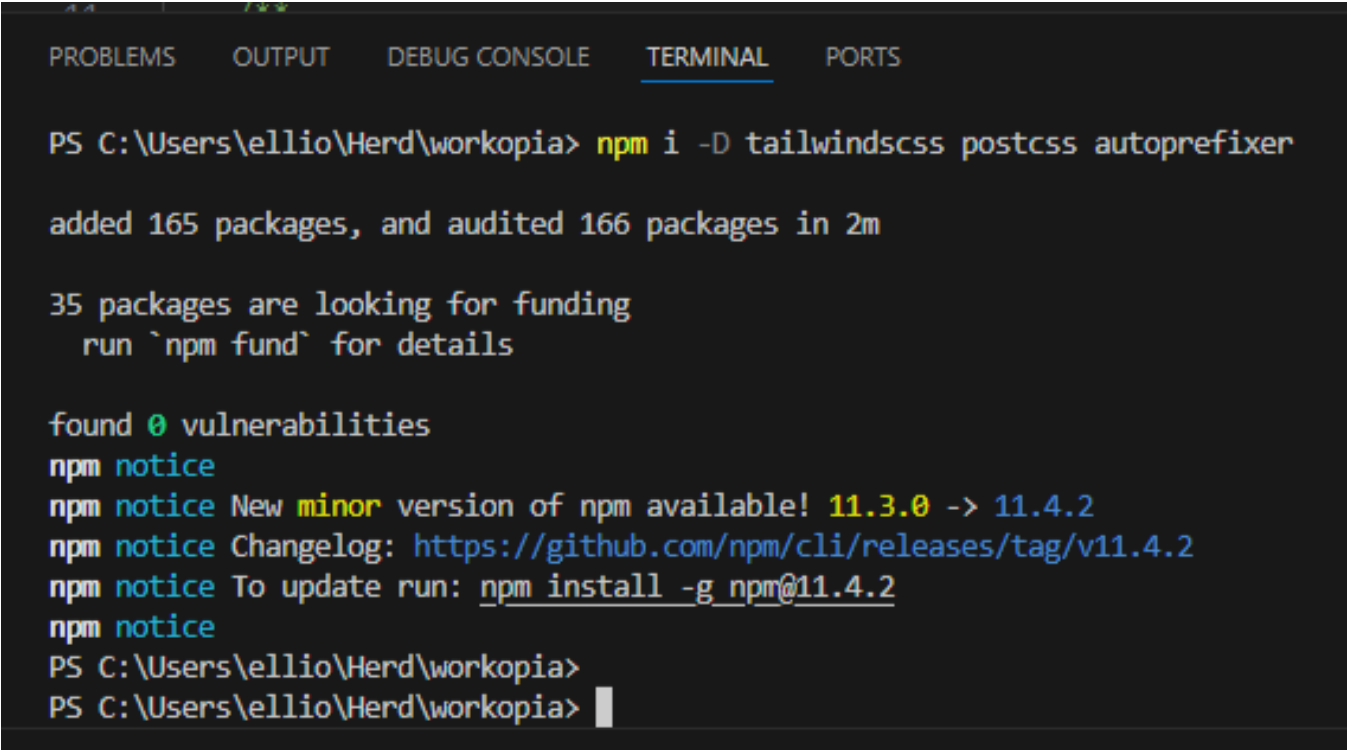
The layout class is modified so that the view points to layout not components.layout.

4.3 Tailwind CSS & Vite Hot Reloading

<https://tailwindcss.com/docs/installation/play-cdn>

Tailwind CSS is a set of utilities for classes. We can use a CDN or NPM to install it. The vite server that is included with Laravel performs hot reloading so that when we change the code the page automatically refreshes.

<https://tailwindcss.com/docs/installation/framework-guides/laravel/vite>

A screenshot of a terminal window with a dark background. At the top, there are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected), and 'PORTS'. The terminal shows the command 'npm i -D tailwindcss postcss autoprefixer' being executed in a PowerShell prompt at 'C:\Users\ellio\Herd\workopia'. The output indicates that 165 packages were added and 166 packages were audited in 2 minutes. It also shows a notice about a new minor version of npm (11.4.2) being available, with a link to the changelog and instructions to update. The prompt returns to 'PS C:\Users\ellio\Herd\workopia>' at the end.

```
PS C:\Users\ellio\Herd\workopia> npm i -D tailwindcss postcss autoprefixer
added 165 packages, and audited 166 packages in 2m

35 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
npm notice
npm notice New minor version of npm available! 11.3.0 -> 11.4.2
npm notice Changelog: https://github.com/npm/cli/releases/tag/v11.4.2
npm notice To update run: npm install -g npm@11.4.2
npm notice
PS C:\Users\ellio\Herd\workopia>
PS C:\Users\ellio\Herd\workopia>
```

From the terminal I use NPM to load three dependencies, 'tailwindcss', 'postcss' and 'autoprefixer'.

Followed the rest of the instructions on the framework guide and now tailwindcss is working

File Edit Selection View Go Run ...

workopia

EXPLORER

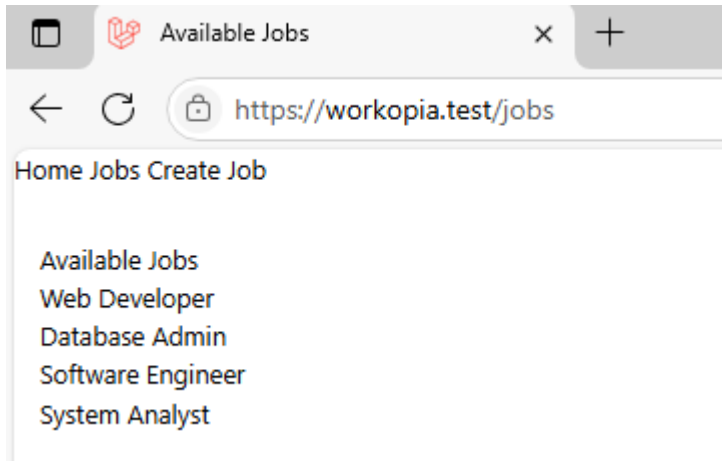
WORKOPIA

- storage
 - app
 - framework
 - logs
 - tests
 - vendor
- .editorconfig
- .env
- .env.example
- .gitattributes
- .gitignore
- .rnd
- artisan
- composer.json
- composer.lock
- package-lock.json
- package.json**
- phpunit.xml
- README.md

blade.php ...pages index.blade.php ...jobs JobController.php crea

```
{ } package.json > ...
1  {
2      "$schema": "https://json.schemastore.org/package.json",
3      "private": true,
4      "type": "module",
5      "scripts": {
6          "build": "vite build",
7          "dev": "vite"
8      },
9      "devDependencies": {
10         "@tailwindcss/vite": "^4.0.0",
11         "autoprefixer": "^10.4.21",
12         "axios": "^1.8.2",
13         "concurrently": "^9.0.1",
14         "laravel-vite-plugin": "^1.2.0",
15         "postcss": "^8.5.6",
16         "tailwindcss": "^4.0.0",
17         "tailwindcss": "^0.3.0",
18         "vite": "^6.2.4"
19     }
20 }
```

Verify the dependencies



Followed the rest of the instructions on the framework guide and now tailwinds css is working.

4.4 Header Component & URL Helper

On the following pages, I am going to build the UI header component.




```
<!-- Mobile Menu -->
<div
  id="mobile-menu"
  class="hidden md:hidden bg-blue-900 text-white mt-5 pb-4 space-y-2">
  <a href="{{url('/jobs')}}" class="block px-4 py-2 hover:bg-blue-700">All Jobs</a>
  <a href="{{url('/jobs/saved')}}" class="block px-4 py-2 hover:bg-blue-700"
    >Saved Jobs</a>
  <a href="{{url('/dashboard')}}" class="block px-4 py-2 hover:bg-blue-700"
    >Dashboard</a>
  <a href="{{url('/login')}}" class="block px-4 py-2 hover:bg-blue-700">Login</a>
  <a href="{{url('/register')}}" class="block px-4 py-2 hover:bg-blue-700"
    >Register</a>
  <a href="{{url('/jobs/create')}}"
    class="block px-4 py-2 bg-yellow-500 hover:bg-yellow-600 text-black">
    <i class="fa fa-edit"></i> Create Job
  </a>
</div>
</header>
```

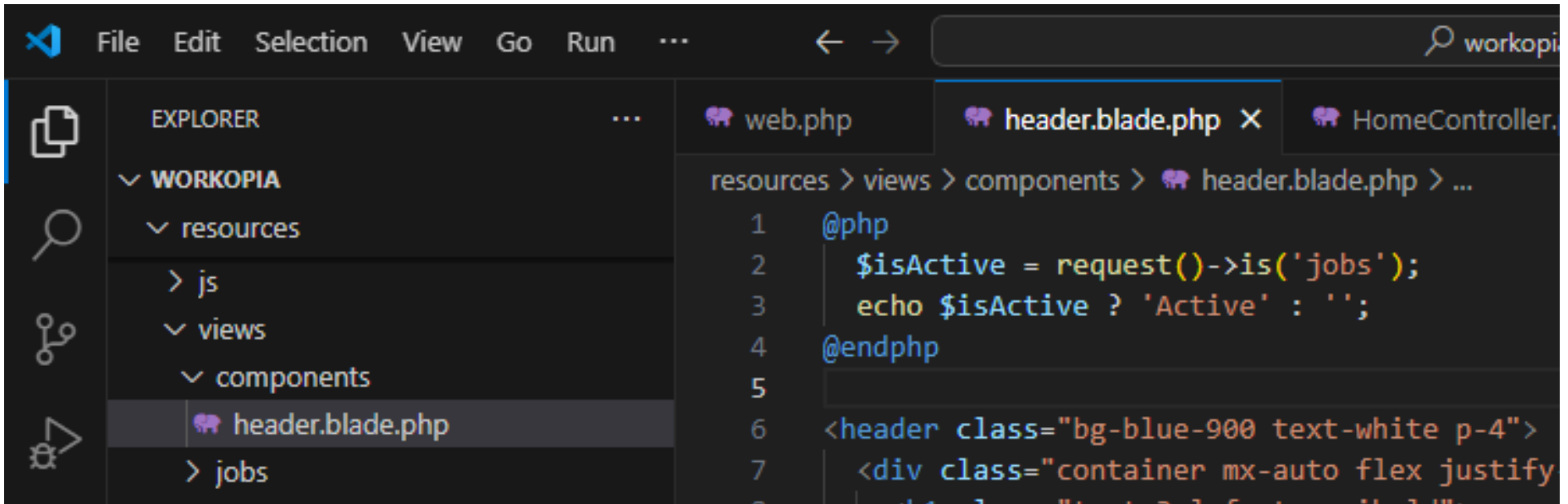
The above code block is for the header.blade.php ui component and is standard html for a header bar with menu links. It contains the tailwind css classes for the relevant styling.

Note in green that I am using the **url helper** so that if the project is in a local dev environment the links will still route correctly. The links will also route correctly if the project was running on a production server.

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.7.2/css/all.min.css" integrity="sh
8   @vite('resources/css/app.css')
9   <title>{{ $title ?? 'Workopia | find and list jobs' }}</title>
10 </head>
11
12 <body class="bg-grey-100">
13   <x-header/>
14
15   <main class="container-mx-auto p-4 mt-4">
16     {{ $slot }}
17   </main>
18 </body>
19
20 </html>
```

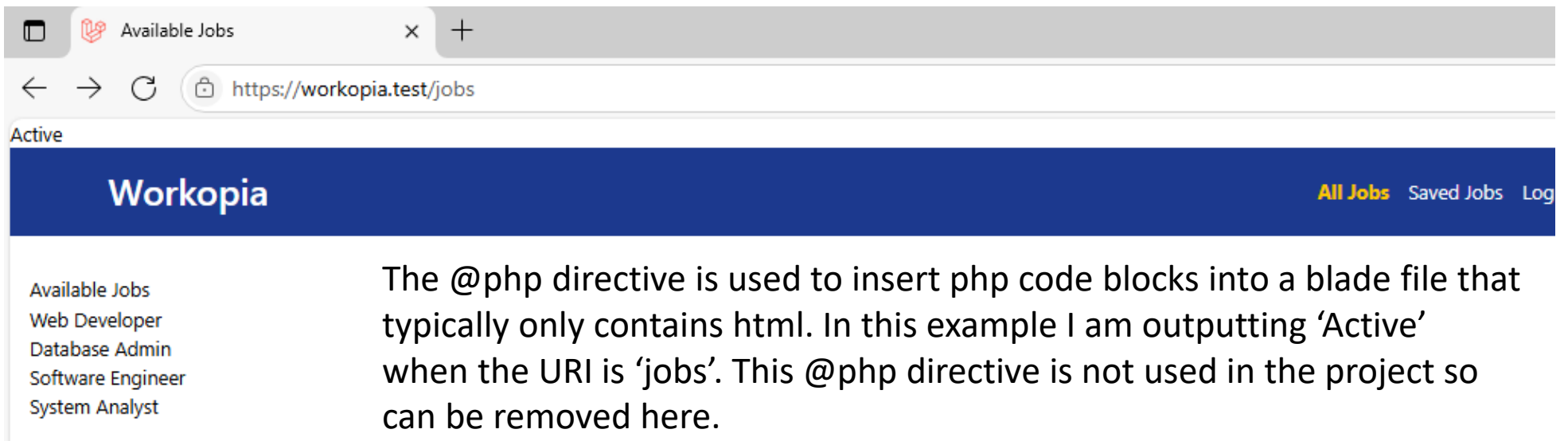
From [Libraries - cdnjs](#) - The #1 free and open source CDN built to make life easier for developers **cdn js, I am going to copy in the tag for the font awesome CDN and put it above the vite css**

4.5 Conditional Classes & @php Directive



The screenshot shows the Visual Studio Code editor interface. The Explorer sidebar on the left displays the project structure: WORKOPIA > resources > views > components > header.blade.php. The main editor area shows the content of header.blade.php, which includes an @php directive used to check if the current request is for the 'jobs' page. The code is as follows:

```
1 @php
2     $isActive = request()->is('jobs');
3     echo $isActive ? 'Active' : '';
4 @endphp
5
6 <header class="bg-blue-900 text-white p-4">
7     <div class="container mx-auto flex justify
```



The screenshot shows a web browser window with the URL <https://workopia.test/jobs>. The page title is 'Available Jobs'. The main content area has a blue header with the 'Workopia' logo and navigation links for 'All Jobs', 'Saved Jobs', and 'Log'. Below the header, there is a list of available jobs:

- Available Jobs
- Web Developer
- Database Admin
- Software Engineer
- System Analyst

The @php directive is used to insert php code blocks into a blade file that typically only contains html. In this example I am outputting 'Active' when the URI is 'jobs'. This @php directive is not used in the project so can be removed here.

```

<a href="{{ url('/') }}" @class([
    'text-white hover:underline py-2' ,
    'font-bold text-yellow-400'=>request()->is('/')
])>Workopia</a>
</h1>
<nav class="hidden md:flex items-center space-x-4">
    <a href="{{ url('/jobs') }}" @class([
        'text-white hover:underline py-2' ,
        'font-bold text-yellow-400'=> request()->is('jobs')
    ])>All Jobs</a>
    <a href="{{ url('/jobs/saved') }}" @class([
        'text-white hover:underline py-2' ,
        'font-bold text-yellow-400'=>request()->is('jobs/saved')
    ])>Saved Jobs</a>
    <a href="{{ url('/login') }}" @class([
        'text-white hover:underline py-2' ,
        'font-bold text-yellow-400'=> request()->is('login')
    ])>Login</a>
    <a href="{{ url('/register') }}" @class([
        'text-white hover:underline py-2' ,
        'font-bold text-yellow-400'=>request()->is('register')
    ])>Register</a>
    <a href="{{ url('/dashboard') }}" @class([
        'text-white hover:underline py-2' ,
        'font-bold text-yellow-400'=>request()->is('dashboard')
    ])>
        <i class="fa fa-gauge mr-1"></i> Dashboard
    </a>

```

Here I am using an **@class** directive to add a text white hover underline class. Additionally, if the link is active, i.e. the **request()->is()**, then I am also adding a **text yellow and bold class**.

```
<a href="{{ url('/') }}"
    class="text-white hover:underline py-2
    {{ request()->is('/') ? 'text-yellow-400 font-bold' : '' }}">Home</a>

<a href="{{ url('/jobs') }}"
    class="text-white hover:underline py-2
    {{ request()->is('jobs') ? 'text-yellow-400 font-bold' : '' }}">All Jobs</a>

<a href="{{ url('/jobs/saved') }}"
    class="text-white hover:underline py-2
    {{ request()->is('jobs/saved') ? 'text-yellow-400 font-bold' : '' }}">Saved Jobs</a>

<a href="{{ url('/login') }}"
    class="text-white hover:underline py-2
    {{ request()->is('login') ? 'text-yellow-400 font-bold' : '' }}">Login</a>

<a href="{{ url('/register') }}"
    class="text-white hover:underline py-2
    {{ request()->is('register') ? 'text-yellow-400 font-bold' : '' }}">Register</a>

<a href="{{ url('/dashboard') }}"
    class="text-white hover:underline py-2
    {{ request()->is('dashboard') ? 'text-yellow-400 font-bold' : '' }}">
    <i class="fa fa-gauge mr-1"></i> Dashboard
</a>
```

I can achieve the same by using a **ternary statement** instead of an @class directive

Available Jobs

Web Developer

Database Admin

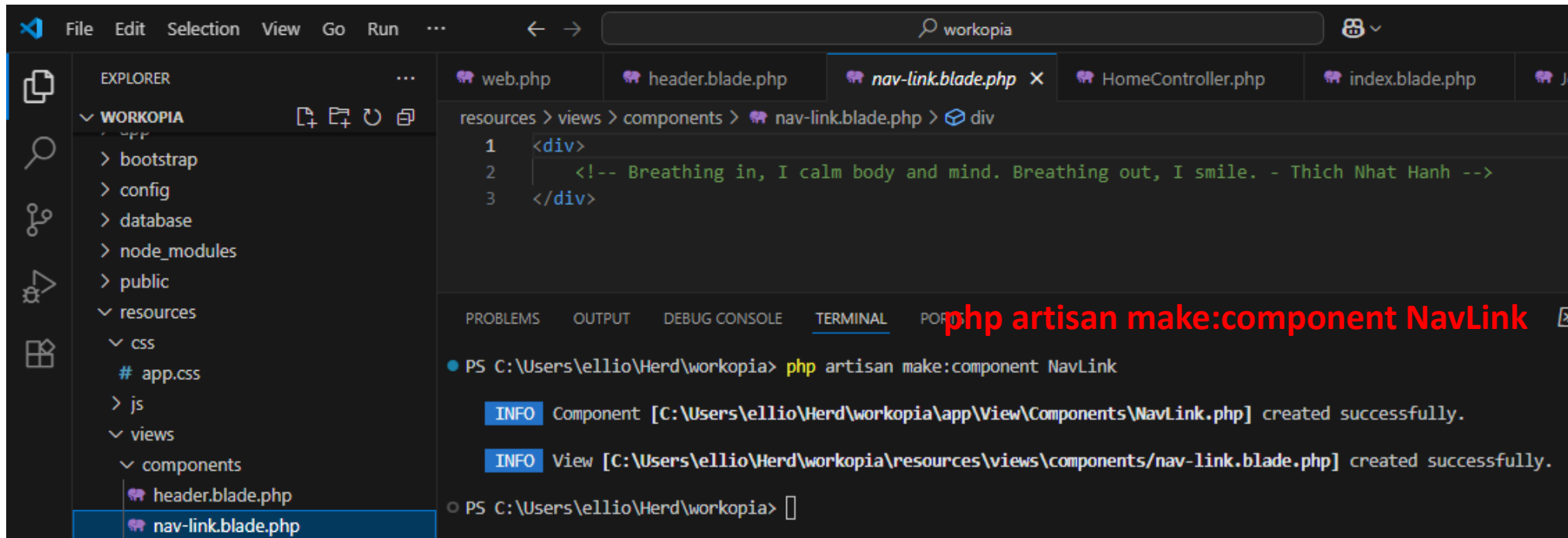
Software Engineer

System Analyst

Welcome to the HOMEPAGE

The conditional class using the ternary statement or the @class directive means that the active nav menu link is highlighted yellow and bold, without using JavaScript.

4.6 Component Attributes & Props



The screenshot shows the Visual Studio Code interface with the Explorer panel on the left displaying the project structure. The file `nav-link.blade.php` is selected under `resources > views > components`. The main editor shows the content of `nav-link.blade.php`, which contains a single `<div>` tag with a comment: `<!-- Breathing in, I calm body and mind. Breathing out, I smile. - Thich Nhat Hanh -->`. The TERMINAL panel at the bottom shows the command `php artisan make:component NavLink` being executed, with two informational messages: "Component [C:\Users\ellio\Herd\workopia\app\View\Components\NavLink.php] created successfully." and "View [C:\Users\ellio\Herd\workopia\resources\views\components\nav-link.blade.php] created successfully."

```
resources > views > components > nav-link.blade.php > div
1 <div>
2     <!-- Breathing in, I calm body and mind. Breathing out, I smile. - Thich Nhat Hanh -->
3 </div>
```

php artisan make:component NavLink

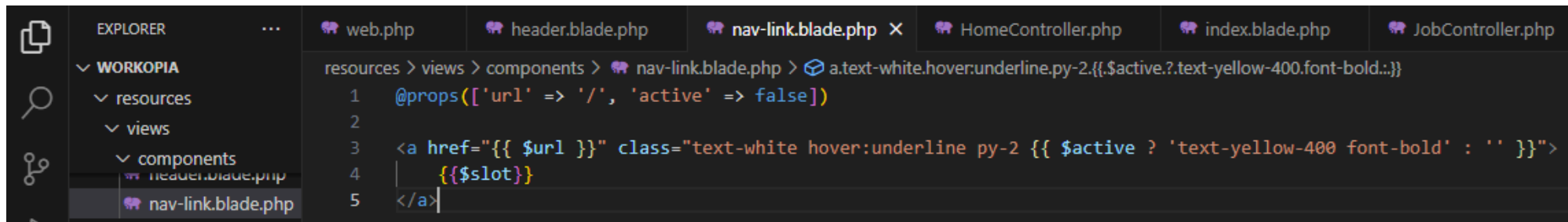
PS C:\Users\ellio\Herd\workopia> php artisan make:component NavLink

INFO Component [C:\Users\ellio\Herd\workopia\app\View\Components\NavLink.php] created successfully.

INFO View [C:\Users\ellio\Herd\workopia\resources\views\components\nav-link.blade.php] created successfully.

PS C:\Users\ellio\Herd\workopia>

I am creating a `nav-link.blade.php` so that I can auto generate the nav-links using props and attributes. The props are an array of custom variables that I am using in the blade. In this case the 'url' is a default value of '/'. The active variable is default set to false.



The screenshot shows the Visual Studio Code interface with the Explorer panel on the left. The file `nav-link.blade.php` is selected under `resources > views > components`. The main editor shows the final code for `nav-link.blade.php`, which includes a `@props` directive and an `<a>` tag with dynamic attributes and a slot.

```
resources > views > components > nav-link.blade.php > a.text-white.hover:underline.py-2.{{ $active ? 'text-yellow-400 font-bold' : '' }}
1 @props(['url' => '/', 'active' => false])
2
3 <a href="{{ $url }}" class="text-white hover:underline py-2 {{ $active ? 'text-yellow-400 font-bold' : '' }}">
4     {{ $slot }}
5 </a>
```

```
<x-nav-link url="/" :active="request()->is('/')">Home</x-nav-link>

<x-nav-link url="/jobs" :active="request()->is('jobs')">All Jobs</x-nav-link>

<x-nav-link url="/jobs/saved" :active="request()->is('jobs/saved')">Saved Jobs</x-
nav-link>

<x-nav-link url="/login" :active="request()->is('login')">Login</x-nav-link>

<x-nav-link url="/register" :active="request()->is('register')">Register</x-nav-
link>

<x-nav-link url="/profile" :active="request()->is('profile')"><i class="fa fa-user
mr-1"></i> Profile</x-nav-link>
```

Now in the 'header.blade.php', I can replace the messy html markup by adding <x-nav-link> that fetches the html from the 'nav-link.blade.php'. I am passing into the prop two dynamic variables. The href and an active variable. Note how the active variable is proceeded with a semi-colon. This is to denote that I am not passing in a string but, in this case a Boolean, because the request()->is() returns nothing or Boolean 1. So when the link is active, I am passing in Boolean value of 1 to the prop.

```
@props(['url' => '/', 'active' => false])

<a href="{{ $url }}" class="text-white hover:underline py-2
{{ $active ? 'text-yellow-400 font-bold' : '' }}">
    {{ $slot }}
</a>
```

In the 'nav-link.blade.php', the prop will receive two variables, the 'url' and a Boolean value for 'active'. I can then use these attributes in the slot html to add the href and the conditional class for an active link.

```
<x-nav-link url="/" :active="request()->is('/')">Home</x-nav-link>
```

The slot is effectively, what is inside the <x-nav-link> tags denoted by red. The slot is handled by the @props which assigns the attributes from the 'header.blade.php' <x-nav-link **attributes**> to dynamic variables which can then be used in the html markup. The {{slot}} part only contains the text of the ahref link.

```
@props(['url' => '/', 'active' => false, 'icon' => NULL])

<a href="{{ $url }}" class="text-white hover:underline py-2
{{ $active ? 'text-yellow-400 font-bold' : '' }}">
    @if($icon)
        <i class="fa fa-{{ $icon }}" mr-1"></i>
    @endif
    {{ $slot }}
</a>
```

```
<x-nav-link url="/profile" :active="request()->is('profile')" icon="user">
Profile</x-nav-link>
<x-nav-link url="/dashboard" :active="request()->is('profile')" icon="gauge">
Dashboard</x-nav-link>
```

I can further refine the code to pass in an optional icon attribute to @props of 'nav-link.blade.php'.
I can handle the icon in an if block.

4.7 Button Link Component Challenge

1. Use artisan to create a 'button-link.blade.php'

```
PS C:\Users\ellio\Herd\workopia> php artisan make:component ButtonLink
```

```
INFO Component [C:\Users\ellio\Herd\workopia\app\View\Components\ButtonLink.php] created successfully.
```

```
INFO View [C:\Users\ellio\Herd\workopia\resources\views\components/button-link.blade.php] created successfully.
```

```
PS C:\Users\ellio\Herd\workopia>
```

2. Create the props and attributes in the html of the blade.

```
@props([
    'url' => '/',
    'icon' => null,
    'bgClass' => 'bg-yellow-500',
    'hoverClass' => 'hover:bg-yellow-600',
    'textClass' => 'text-black'
])

<a href="{{ $url }}"
class="{{ $bgClass }} {{ $hoverClass }} {{ $textClass }} px-4 py-2 rounded hover:shadow-md
transition duration-300">
    @if($icon)
        <i class="fa fa-{{ $icon }} mr-1"></i>
    @endif
    {{ $slot }}
</a>
```

3. Use the button-link.blade in the header blade.

```
<header class="bg-blue-900 text-white p-4">
  <div class="container mx-auto flex justify-between items-center">
    <h1 class="text-3xl font-semibold">
      <x-nav-link url="/" :active="request()->is('/')">Workopia</x-nav-link>
    </h1>
    <nav class="hidden md:flex items-center space-x-4">
      <x-nav-link url="/" :active="request()->is('/')">Home</x-nav-link>
      <x-nav-link url="/jobs" :active="request()->is('jobs')">All Jobs</x-nav-link>
      <x-nav-link url="/jobs/saved" :active="request()->is('jobs/saved')">Saved Jobs</x-nav-
link>
      <x-nav-link url="/login" :active="request()->is('login')">Login</x-nav-link>
      <x-nav-link url="/register" :active="request()->is('register')">Register</x-nav-link>
      <x-nav-link url="/profile" :active="request()->is('profile')" icon="user"> Profile</x-
nav-link>
      <x-nav-link url="/dashboard" :active="request()->is('profile')" icon="gauge">
Dashboard</x-nav-link>
      <x-button-link url="/jobs/create" icon="edit">Create Job</x-button-link>
    </nav>
    <button id="hamburger" class="text-white md:hidden flex items-center">
      <i class="fa fa-bars text-2xl"></i>
    </button>
```

4.8 Mobile Menu Nav Link

1. Add an additional prop for 'mobile' to the nav-link.blade with a default of null so that if the 'mobile' attribute is passed in I can render different html for a mobile menu link.

```
@props(['url' => '/', 'active' => false, 'icon' => NULL, 'mobile' => NULL])

@if($mobile)
<a href="{{ $url }}" class="block px-4 py-2 hover:bg-blue-700 {{$active ? 'text-yellow-500 font-bold' : ''}}">
    @if($icon)
    <i class="fa fa-{{$icon}}" mr-1"></i>
    @endif
    {{$slot}}
</a>
@else
<a href="{{ $url }}" class="text-white hover:underline py-2 {{$active ? 'text-yellow-500 font-bold' : ''}}">
    @if($icon)
    <i class="fa fa-{{$icon}}" mr-1"></i>
    @endif
    {{$slot}}
</a>
@endif
</div>
```

2. Add another prop to the button-link.blade so that the buttons on the mobile menu have an additional class of block.

```
@props([
    'url' => '/',
    'icon' => NULL,
    'bgClass' => 'bg-yellow-500',
    'hoverClass' => 'hover:bg-yellow-600',
    'textClass' => 'text-black',
    'block' => FALSE
])

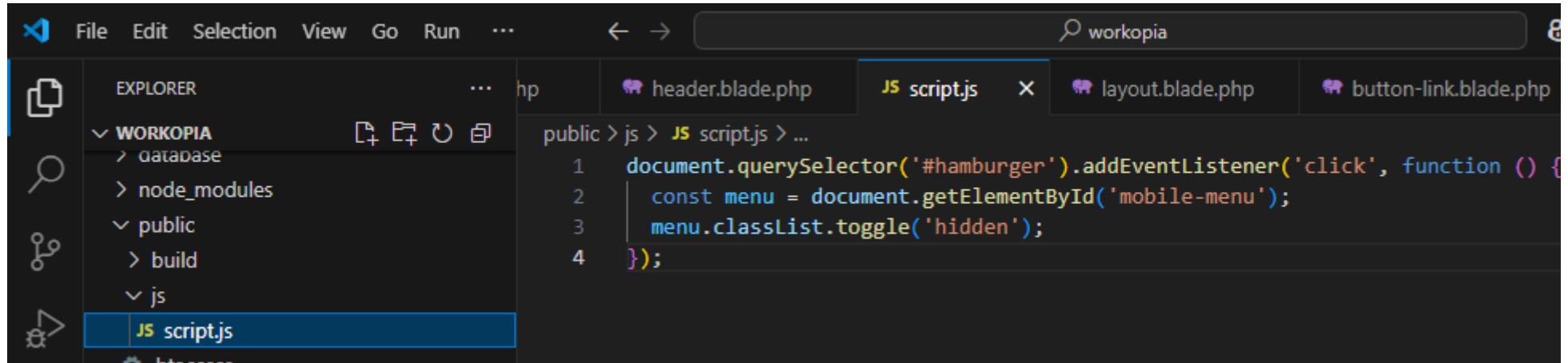
<a href="{{ $url }}"
class="{{{$bgClass}} {{$hoverClass}} {{$textClass}} px-4 py-2 rounded hover:shadow-md transition duration-300
{{{$block ? 'block' : ''}}}">
    @if($icon)
        <i class="fa fa-{{ $icon }}" mr-1"></i>
    @endif
    {{$slot}}
</a>
```

3. Use the nav.link.blade and button-link.blade in the mobile nav menu.

```
<!-- Mobile Menu -->
<div id="mobile-menu" class="hidden md:hidden bg-blue-900 text-white mt-5 pb-4 space-y-2">
    <x-nav-link url="/jobs" :active="request()->is('jobs')" :mobile="true">All Jobs</x-nav-link>
    <x-nav-link url="/jobs/saved" :active="request()->is('jobs/saved')" :mobile="true">Saved Jobs</x-nav-link>
    <x-nav-link url="/login" :active="request()->is('login')" :mobile="true">Login</x-nav-link>
    <x-nav-link url="/register" :active="request()->is('register')" :mobile="true">Register</x-nav-link>
    <x-nav-link url="/dashboard" :active="request()->is('dashboard')" :mobile="true">Dashbaord</x-nav-link>
    <x-button-link url="/jobs/create" :block="true" icon="edit"> Create Job</x-button-link>
</div>
```

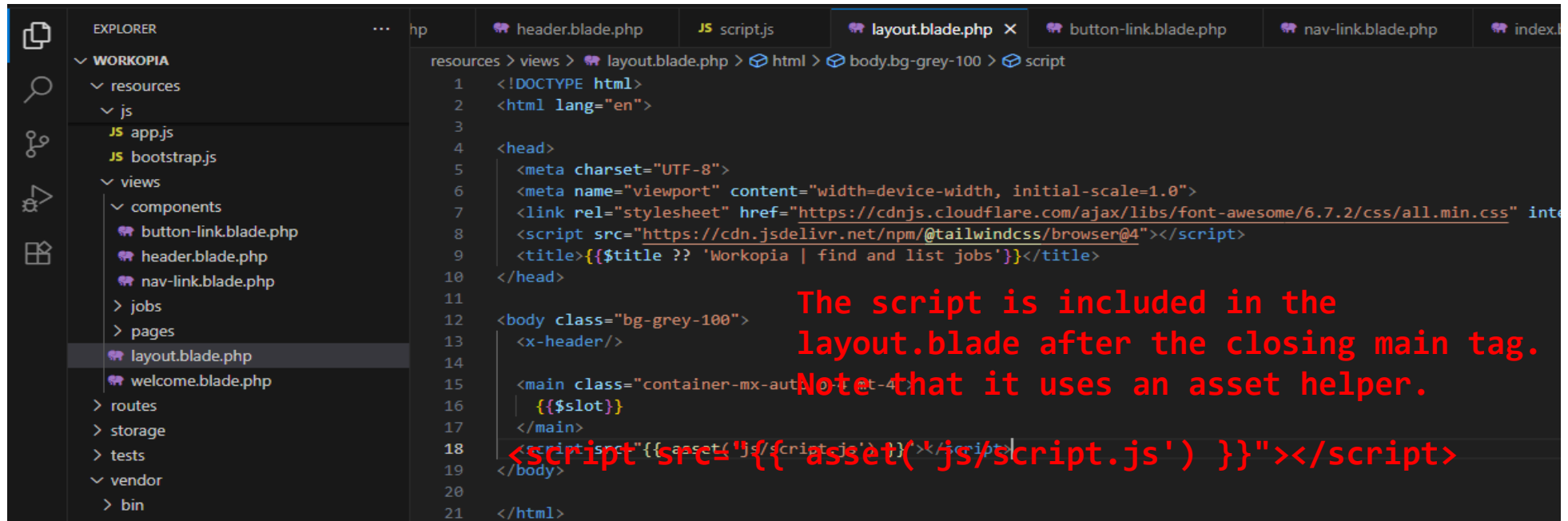

4.9 Mobile Menu Toggle

1. Create a file at `/public/js/script.js` and add some simple vanilla JS to include an event listener for click on the hamburger button to toggle the hidden class on the nav-manu element.



This screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left displays the project structure, with the file `JS script.js` selected under the `public/js` directory. The main editor window shows the content of `script.js`, which contains a JavaScript function to toggle the 'hidden' class on the 'mobile-menu' element when the 'hamburger' button is clicked.

```
1 document.querySelector('#hamburger').addEventListener('click', function () {
2     const menu = document.getElementById('mobile-menu');
3     menu.classList.toggle('hidden');
4 });
```



This screenshot shows the Visual Studio Code interface with the `layout.blade.php` file open in the main editor. The file structure in the Explorer sidebar shows the file is located within the `views` directory. The code in the editor shows the HTML structure of the layout, with the `script.js` file included in the body using the `asset` helper. A red text overlay provides additional context.

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5     <meta charset="UTF-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/@tailwindcss/browser@4"></script>
8     <script src="https://cdn.jsdelivr.net/npm/@tailwindcss/browser@4"></script>
9     <title>{{ $title ?? 'Workopia | find and list jobs' }}</title>
10 </head>
11
12 <body class="bg-grey-100">
13     <x-header/>
14
15     <main class="container-mx-auto">
16         {{ $slot }}
17     </main>
18     <script src="{{ asset('js/script.js') }}"></script>
19 </body>
20
21 </html>
```

The script is included in the layout.blade after the closing main tag. Note that it uses an asset helper.

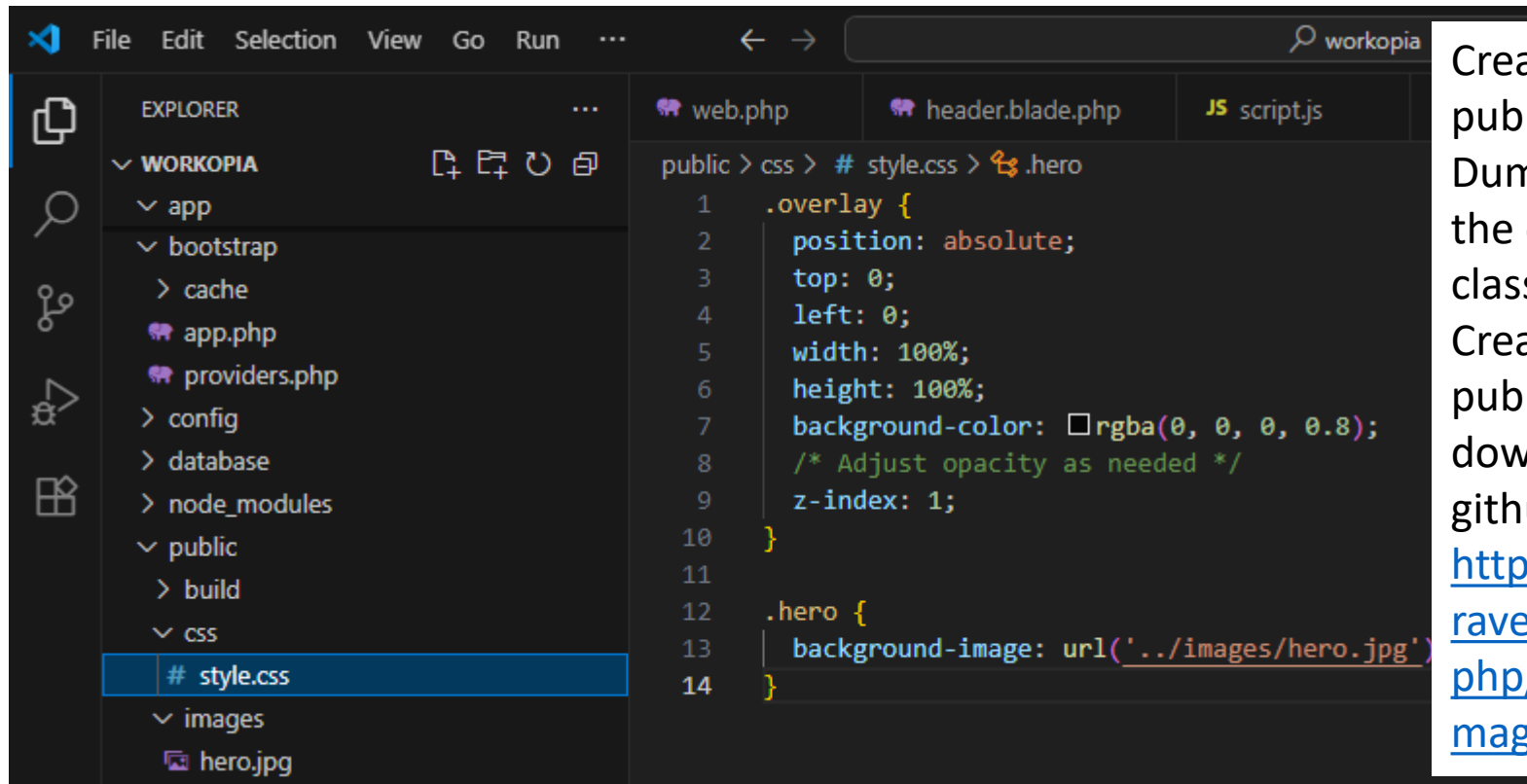
4.10 Hero Component

```
PS C:\Users\ellio\Herd\workopia> php artisan make:component Hero
```

```
INFO Component [C:\Users\ellio\Herd\workopia\app\View\Components\Hero.php] created successfully.
```

```
INFO View [C:\Users\ellio\Herd\workopia\resources\views\components\hero.blade.php] created successfully.
```

```
PS C:\Users\ellio\Herd\workopia>
```



Create folder and file public/css/style.css.
Dump in the css code for the overlay and hero classes.
Create folder public/images and download the img from github
<https://github.com/bradtraversy/workopia-php/blob/main/public/images/showcase.jpg>

```
File Edit Selection View Go Run ... workopia
```

EXPLORER

- WORKOPIA
 - resources
 - views
 - components
 - hero.blade.php
 - nav-link.blade.php
 - jobs
 - pages
 - layout.blade.php
 - welcome.blade.php
 - routes
 - storage
 - tests
 - vendor
 - .editorconfig
 - .env
 - .env.example
 - .gitattributes
 - .gitignore

resources > views > layout.blade.php > html > head > link

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/@tailwindcss/browser@4"></script>
8   <script src="https://cdn.jsdelivr.net/npm/@tailwindcss/browser@4"></script>
9   <link rel="stylesheet" href="{{ asset('css/style.css') }}" />
10  <title>{{ $title ?? 'Workopia | find and list jobs' }}</title>
11  </head>
12
13 <body class="bg-grey-100">
14   <x-header/>
15   @if(request()->is('/'))
16   <x-hero/>
17   @endif
18   <main class="container-mx-auto p-4 mt-4">
19     {{ $slot }}
20   </main>
21   <script src="{{ asset('js/script.js') }}"></script>
```

<link rel="stylesheet" href="{{ asset('css/style.css') }}" />

@if(request()->is('/'))
<x-hero/>
@endif

Add the stylesheet into the header of the layout.blade.

In the layout blade and an @if block to only load the hero.blade on the homepage.

```
@props(['title' => 'Find Your Dream Job'])
```

```
<section
  class="hero relative bg-cover bg-center bg-no-repeat h-72 flex items-center"
>
  <div class="overlay"></div>
  <div class="container mx-auto text-center z-10">
    <h2 class="text-4xl text-white font-bold mb-4">{{ $title }}</h2>
    <form class="block mx-5 md:mx-auto md:space-x-2">
      <input
        type="text"
        name="keywords"
        placeholder="Keywords"
        class="w-full md:w-72 px-4 py-3 focus:outline-none"
      />
      <input
        type="text"
        name="location"
        placeholder="Location"
        class="w-full md:w-72 px-4 py-3 focus:outline-none"
      />
      <button
        class="w-full md:w-auto bg-blue-700 hover:bg-blue-600 text-white px-4 py-3
focus:outline-none"
      >
        <i class="fa fa-search mr-1"></i> Search
      </button>
    </form>
  </div>
</section>
```

**Build the hero
Blade and add a
prop to take an
optional title string**

4.11 Top & Bottom banner

```
PS C:\Users\ellio\Herd\workopia> php artisan make:component Topbanner
```

```
INFO Component [C:\Users\ellio\Herd\workopia\app\View\Components\Topbanner.php] created successfully.
```

```
INFO View [C:\Users\ellio\Herd\workopia\resources\views\components/topbanner.blade.php] created successfully.
```

```
PS C:\Users\ellio\Herd\workopia> php artisan make:component BottomBanner
```

```
INFO Component [C:\Users\ellio\Herd\workopia\app\View\Components\BottomBanner.php] created successfully.
```

```
INFO View [C:\Users\ellio\Herd\workopia\resources\views\components/bottom-banner.blade.php] created successfully.
```

```
PS C:\Users\ellio\Herd\workopia>
```

```
@props([
    'heading' => 'Unlock Your Career Potential',
    'subheading' => 'Discover the perfect job opportunity for you'
])
```

```
<section class="bg-blue-900 text-white py-6 text-center">
    <div class="container mx-auto">
        <h2 class="text-3xl font-semibold">
            {{ $heading }}
        </h2>
        <p class="text-lg mt-2">
            {{ $subheading }}
        </p>
    </div>
</section>
```

Create top and bottom banner blades.

In the layout blade I only want the top-banner on the homepage.

```
@if(request()->is('/'))
    <x-hero/>
    <x-top-banner />
@endif
```

```
@props([
    'heading' => 'Looking to hire?',
    'subheading' => 'Post your job listing now and find the perfect candidate'
])
```

```
<section class="container mx-auto my-6">
    <div class="bg-blue-800 text-white rounded p-4 flex items-center justify-between">
        <div>
            <h2 class="text-xl font-semibold">{{ $heading }}</h2>
            <p class="text-gray-200 text-lg mt-2">
                {{ $subheading }}
            </p>
        </div>
        <x-button-link url="/jobs/create" type="button" icon="edit">Create Job</x-button-link>
    </div>
</section>
```

The bottom banner can go in the pages/index.blade

```
<x-layout>
    <h2>Welcome to the HOMEPAGE</h2>
    <x-bottom-banner/>
</x-layout>
```

Find Your Dream Job

Location

 Search

Unlock Your Career Potential

Discover the perfect job opportunity for you

Welcome to the HOMEPAGE

Looking to hire?

Post your job listing now and find the perfect candidate

[Create Job](#)

5. PostgreSQL Database Setup & Migrations

[5.1 Database options](#)

[5.2 Create Database & user](#)

[5.3 Configure Database connection](#)

[5.4 Migrations Overview & Commands](#)

[5.5 Creating Migrations](#)

5.1 Database Options



Laravel From Scratch

Database & Migrations - Section Intro

- ✓ **Database Options**
- ✓ **PostgreSQL Install**
- ✓ **PG Admin Tool**
- ✓ **Create Database & User**
- ✓ **Configure Connection**
- ✓ **Migrations Overview**
- ✓ **Create & Run Migrations**

Databases

Laravel Supports multiple databases out of the box

✓ **SQLite**

✓ **MySQL**

✓ **PostgreSQL**

✓ **SQL Server**

✓ **MariaDB**

✓ **Oracle**

✓ **MongoDB**

✓ **Redis**

✓ **Neo4J**

✓ **InfluxDB**

Databases Config

.env File

PostgreSQL

```
DB_CONNECTION=pgsql
DB_HOST=127.0.0.1
DB_PORT=5432
DB_DATABASE=your_database_name
DB_USERNAME=your_database_user
DB_PASSWORD=your_database_password
```

MySQL

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=your_database_name
DB_USERNAME=your_database_user
DB_PASSWORD=your_database_password
```

SQLite

```
DB_CONNECTION=sqlite
# DB_HOST=127.0.0.1
# DB_PORT=3306
# DB_DATABASE=
# DB_USERNAME=
# DB_PASSWORD=
```



EXPLORER

WORKOPIA

resources

views

layout.blade.php

welcome.blade.php

routes

storage

tests

vendor

.editorconfig

.env

.env.example

.gitattributes

.gitignore

.rnd

artisan

composer.json

composer.lock

package-lock.json

package.json

layout.blade.php

hero.blade.php

topbanner.b

.env

14 PHP_CLI_SERVER_WORKERS=4

15

16 BCRYPT_ROUNDS=12

17

18 LOG_CHANNEL=stack

19 LOG_STACK=single

20 LOG_DEPRECATED_CHANNEL=null

21 LOG_LEVEL=debug

22

23 DB_CONNECTION=sqlite

24 # DB_HOST=127.0.0.1

25 # DB_PORT=3306

26 # DB_DATABASE=laravel

27 # DB_USERNAME=root

28 # DB_PASSWORD=

29

30 SESSION_DRIVER=database

31 SESSION_LIFETIME=120

32 SESSION_ENCRYPT=false

33 SESSION_PATH=/

34 SESSION_DOMAIN=null

35

The .env file contains the database connection parameters

5.2 Create PostgreSQL Database & User

```
CREATE DATABASE workopia;
```

```
CREATE USER workopia WITH SUPERUSER PASSWORD 'password';
```

```
GRANT ALL PRIVILEGES ON DATABASE workopia TO workopia;
```

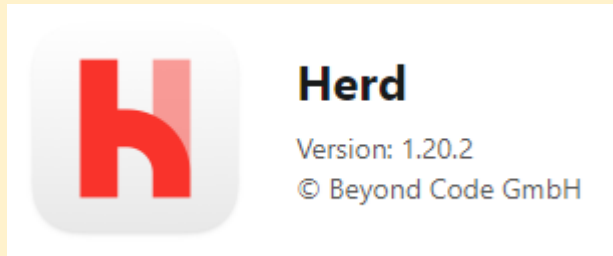
5.3 Configure Database Connection

Windows 11 base machine
running PostgreSQL Server



192.168.1.49

Windows 10 in Virtualbox



192.168.1.50

DB_CONNECTION=pgsql
DB_HOST=192.168.1.49
DB_PORT=5432
DB_DATABASE=workopia
DB_USERNAME=workopia
DB_PASSWORD=password

Postgresql.conf → ensure listening address is *

listen_addresses = '*'

Pghba.conf → add a rule to allow connections from any LAN IP address

host all Phalcon 192.168.1.0/0 md5

Add a firewall rule in the windows 11 machine to allow port 5432 as per:

<https://blog.devart.com/configure-postgresql-to-allow-remote-connection.html>

**Fettling to get
the db
connection to
work because
the db is on a
different host.**

Internal Server Error

`Illuminate\Database\QueryException`

SQLSTATE[42P01]: Undefined table: 7 ERROR:
relation "sessions" does not exist LINE 1: select *
from "sessions" where "id" = \$1 limit 1 ^
(Connection: pgsql, SQL: select * from "sessions"
where "id" =
z6MKCvnTLdteY2BQmwKXskRGdjPHD2uNTaOrQ
8Pe limit 1)

`GET workopia.test`

PHP 8.4.8 — Laravel 12.19.3

This is normal and expected because I have to perform some migrations to get specific tables in the PostgreSQL database so that it can work with Laravel.

Additionally, from vscode terminal, I can use tinker to verify the database connection.

```
PS C:\Users\ellio\Herd\workopia> php artisan tinker
Psy Shell v0.12.8 (PHP 8.4.8 - cli) by Justin Hileman
> DB::select('SELECT version()')
= [
    {#6198
        +"version": "PostgreSQL 17.4 on x86_64-windows, compiled by msvc-19.43.34808, 64-bit",
    },
]
>
```

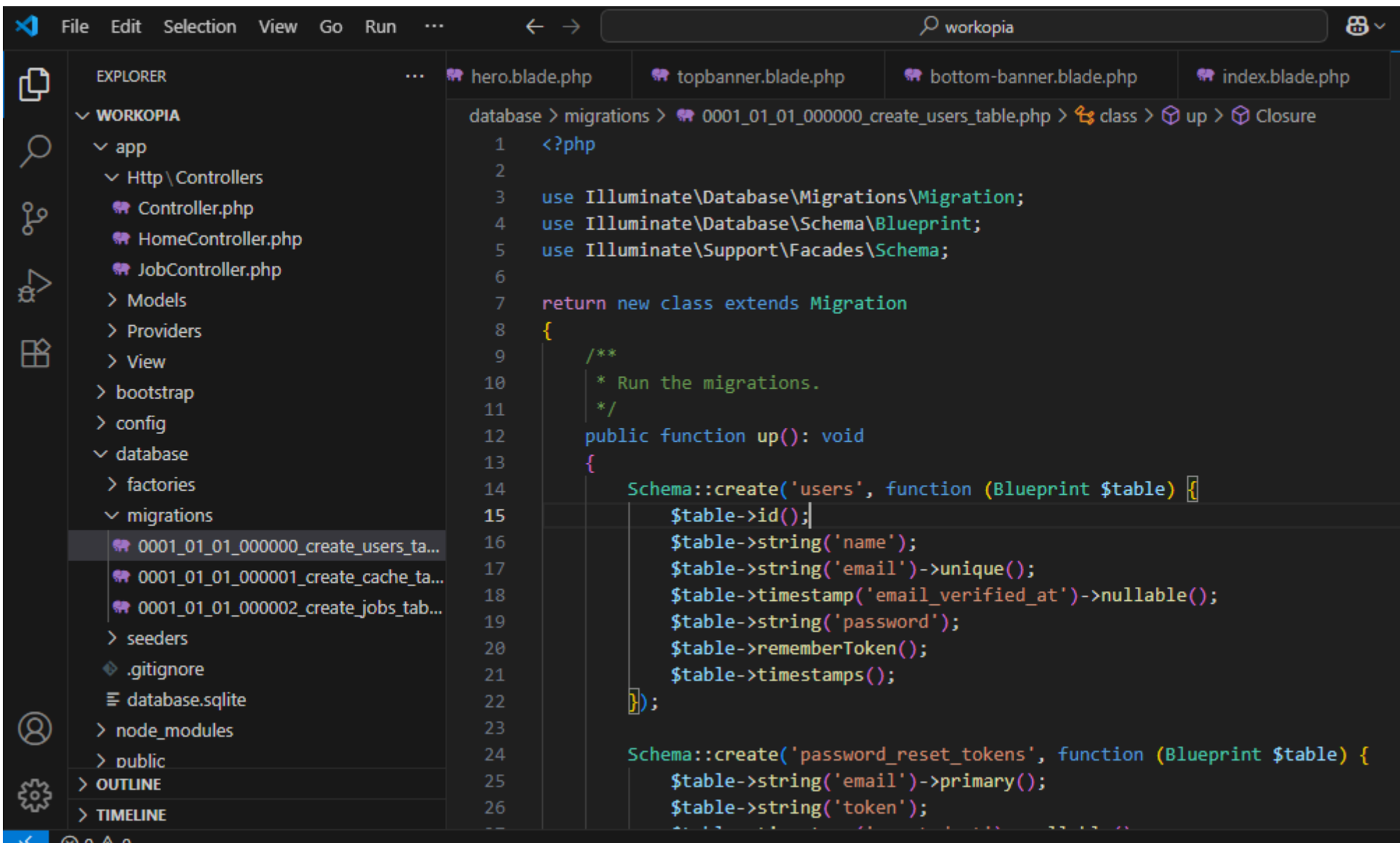
5.4 Migrations Overview & Commands

What Are Migrations?

Migrations are version-controlled files used to manage database schema changes. They allow you to create, modify or delete tables and columns in a structured and repeatable way

Migrations are run from the **database/migrations** folder

php artisan migrate



The database/migrations folder contains three files, users table, cache, and jobs (background jobs) and they are prefixed with timestamps.

The database/migrations files are formatted as a class that extends the migration class and they have two methods up and down. The up function is to create tables and the down function is to undo the up function.

```
database > migrations > 0001_01_01_000000_create_users_table.php > class > up > Closure
7   return new class extends Migration
10       * Run the migrations.
11       */
12   public function up(): void
13   {
14       Schema::create('users', function (Blueprint $table) {
15           $table->id();
16           $table->string('name');
17           $table->string('email')->unique();
18           $table->timestamp('email_verified_at')->nullable();
19           $table->string('password');
20           $table->rememberToken();
21           $table->timestamps();
22       });
```

Looking at the create users function within the up, it has code to create a default user table with fields such as id, name, email (unique) timestamp that the email was verified (optional), password and some token stuff.

Migration Commands

migrate - Runs all of the migrations that are in the migrations directory.
migrate:fresh - Completely drops all tables and re-runs all migrations.
migrate:install - Creates the migrations table.
migrate:refresh - Rolls back all migrations and then re-applies them.
migrate:reset - Rolls back all of the migrations that have been run.
migrate:rollback - Rolls back the last migration that was run.
migrate:status - Shows the status of the migrations.

The migrations command can perform various actions during the development stage of the project. You would not normally want to run these commands in production because you would lose all the data in the database.

```
PS C:\Users\ellio\Herd\workopia> php artisan migrate
```

```
INFO  Preparing database.
```

```
Creating migration table ..... 200.88ms DONE
```

```
INFO  Running migrations.
```

```
0001_01_01_000000_create_users_table ..... 379.55ms DONE
```

```
0001_01_01_000001_create_cache_table ..... 26.91ms DONE
```

```
0001_01_01_000002_create_jobs_table ..... 105.36ms DONE
```

```
PS C:\Users\ellio\Herd\workopia>
```

The php artisan migrate command generates the necessary starter tables for our Laravel web app to work.

▼  Tables (9)

>  cache

>  cache_locks

>  failed_jobs

>  job_batches

>  jobs

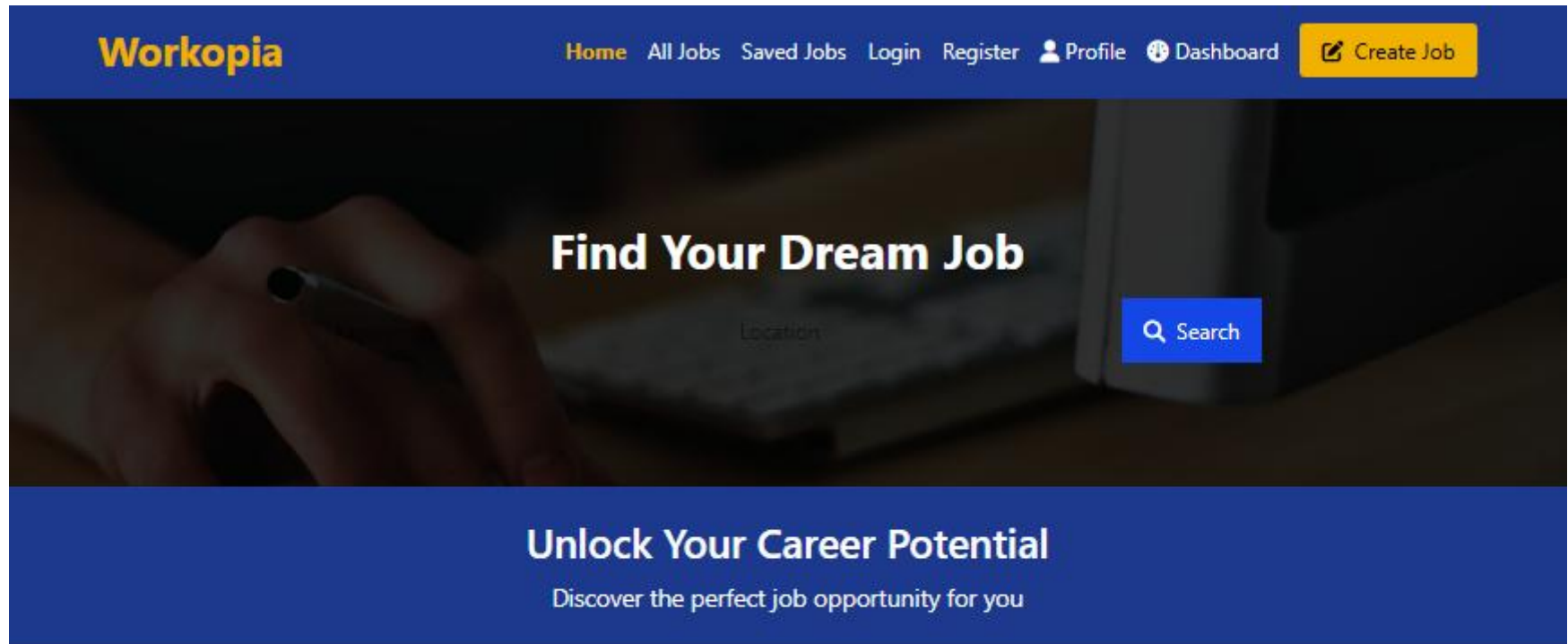
>  migrations

>  password_reset_tokens

>  sessions

>  users

Now that the migrations starter tables are in the database, the Laravel app is working correctly. Note that I have not actually created any tables as such, these are system tables for Laravel.



Welcome to the HOMEPAGE

Looking to hire?

Post your job listing now and find the perfect candidate

 **Create Job**

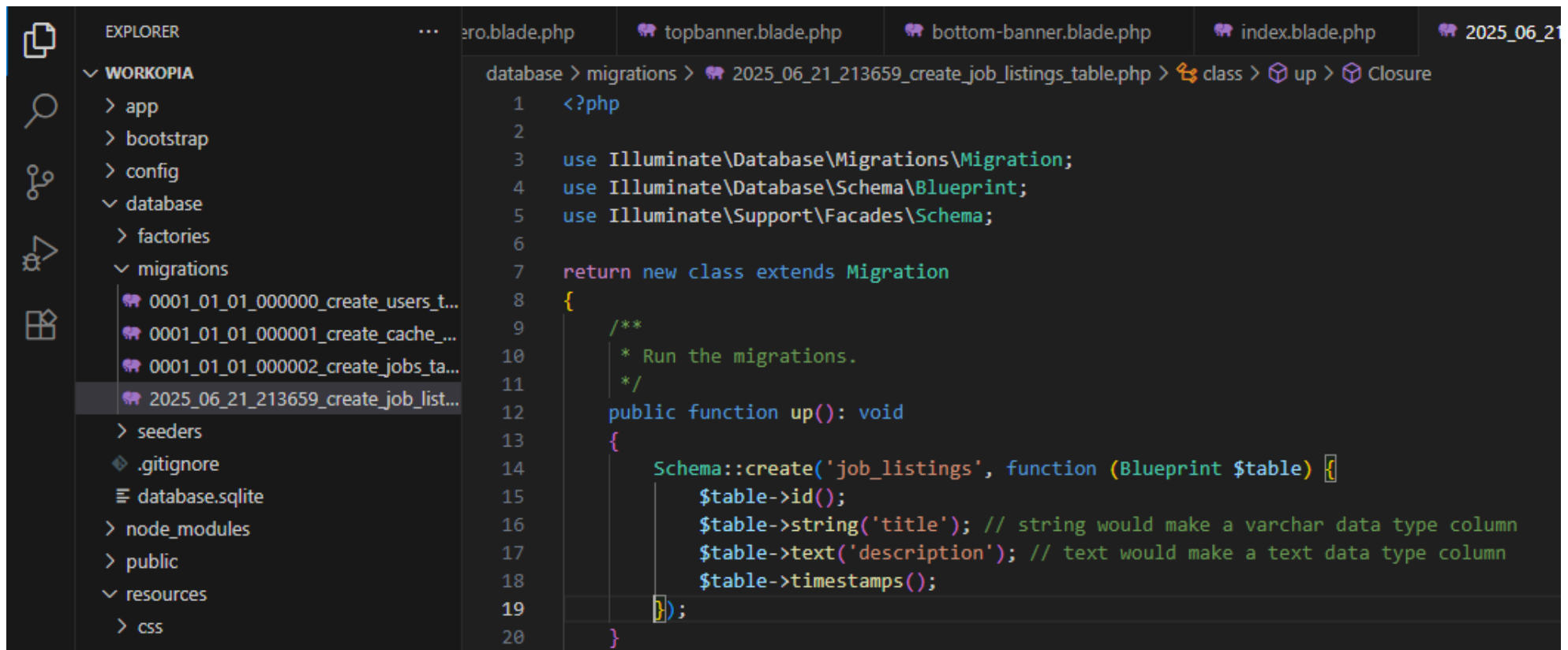
5.5 Create Migrations

Whenever we want to create or modify a database table, we create a new migration file so that there is versioning of the database structure within the project. I use artisan to create a new migration.

```
PS C:\Users\ellio\Herd\workopia> php artisan make:migration create_job_listings_table

INFO  Migration
[C:\Users\ellio\Herd\workopia\database\Migrations\2025_06_21_213659_create_job_listings_table.php] created successfully.

PS C:\Users\ellio\Herd\workopia>
```



```
database > migrations > 2025_06_21_213659_create_job_listings_table.php > class > up > Closure
1  <?php
2
3  use Illuminate\Database\Migrations\Migration;
4  use Illuminate\Database\Schema\Blueprint;
5  use Illuminate\Support\Facades\Schema;
6
7  return new class extends Migration
8  {
9      /**
10       * Run the migrations.
11       */
12     public function up(): void
13     {
14         Schema::create('job_listings', function (Blueprint $table) {
15             $table->id();
16             $table->string('title'); // string would make a varchar data type column
17             $table->text('description'); // text would make a text data type column
18             $table->timestamps();
19         });
20     }
```

The make:migration command generates a migrations template file where we can define the columns of our table. Now I can save the modified migration file and run it from artisan.

```
PS C:\Users\ellio\Herd\workopia> php artisan migrate
```

```
INFO  Running migrations.
```

```
2025_06_21_213659_create_job_listings_table ..... 11.46ms DONE
```

- > FTS Dictionaries
- > Aa FTS Parsers
- > FTS Templates
- > Foreign Tables
- > Functions
- > Materialized Views
- > Operators
- > Procedures
- > 1.3 Sequences
- ▼ Tables (10)
 - > cache
 - > cache_locks
 - > failed_jobs
 - > job_batches
 - > job_listings
 - > jobs
 - > migrations
 - > password_reset_tokens
 - > sessions
 - > users
- > Trigger Functions
- > Types
- > Views

job_listings

GeneralColumnsAdvancedConstraintsPartitionsParametersSecuritySQL

Inherited from table(s)

Columns

| | Name | Data type | Length/Precision | Scale | Not NULL? | Primary key? | Default |
|--|-------------|-------------------|------------------|-------|-------------------------------------|-------------------------------------|------------|
| | id | bigint | | | ☒ | ☒ | nextval('j |
| | title | character varying | 255 | | ☒ | ☐ | |
| | description | text | | | ☒ | ☐ | |
| | created_at | timestamp without | 0 | | ☐ | ☐ | |
| | updated_at | timestamp without | 0 | | ☐ | ☐ | |

In the database I can see that the table has been create with all the necessary fields. Note that because the migration files are timestamped, it did not re-run the three tables from the first migration.

6. Models, Eloquent ORM, Factories & Seeders

[6.1 Intro to Models](#)

[6.2 Fetching Data & Eloquent ORM](#)

[6.3 Tinker & CRUD Operations](#)

[6.4 Model Binding & Single Job Listing](#)

[6.5 Create Job Listing](#)

[6.6 Input Validation & Errors](#)

[6.7 Job Schema Update Migration](#)

[6.8 Eloquent Relationships](#)

[6.9 Using Factories](#)

[6.10 Creating Factory & Faker Library](#)

[6.11 Creating Seeders](#)

[6.12 Final Database Seeder](#)

[6.13 Note for MySQL/MariaDb Users](#)

6.1 Intro to Models



Laravel From Scratch

Models, Eloquent, Factories, Seeders - Section Intro

✓ **Intro To Models**

✓ **Eloquent ORM**

✓ **Tinker REPL**

✓ **Input Validation**

✓ **Model Relationships**

✓ **Factories & Faker**

✓ **Database Seeding**

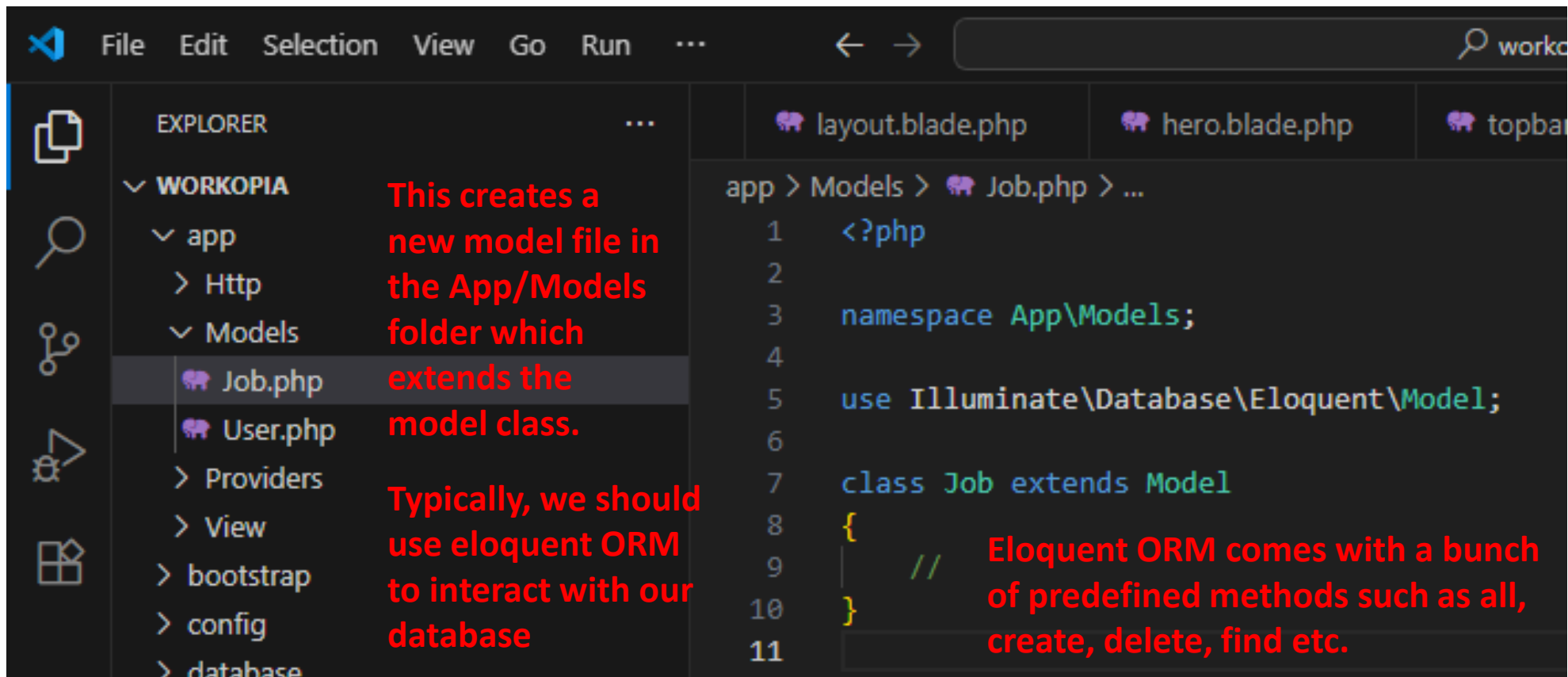
6.2 Fetching Data & Eloquent ORM

```
PS C:\Users\ellio\Herd\workopia> php artisan make:model Job --factory
```

```
INFO Model [C:\Users\ellio\Herd\workopia\app\Models\Job.php] created successfully.
```

```
INFO Factory [C:\Users\ellio\Herd\workopia\database\factories\JobFactory.php] created successfully.
```

```
PS C:\Users\ellio\Herd\workopia>
```



EXPLORER

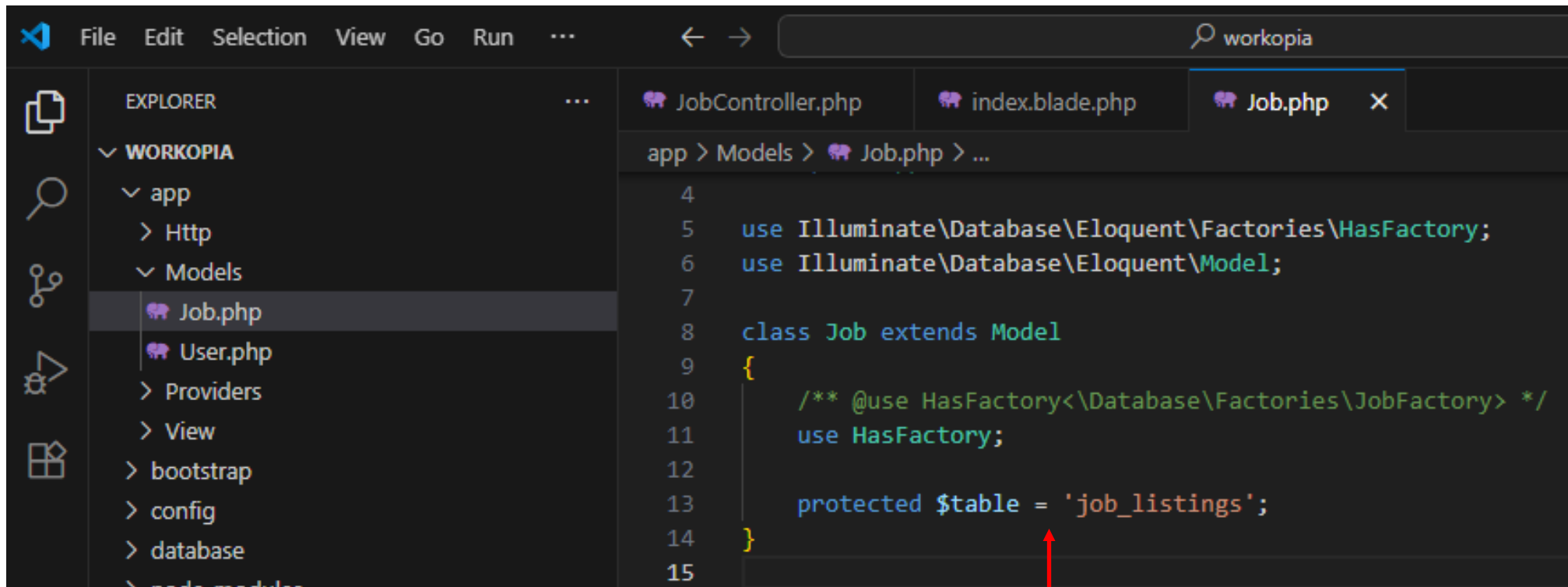
- WORKOPIA
 - app
 - Http
 - Models
 - Job.php**
 - User.php
 - Providers
 - View
 - bootstrap
 - config
 - database

This creates a new model file in the App/Models folder which extends the model class.

Typically, we should use eloquent ORM to interact with our database

Eloquent ORM comes with a bunch of predefined methods such as all, create, delete, find etc.

```
app > Models > Job.php > ...
1  <?php
2
3  namespace App\Models;
4
5  use Illuminate\Database\Eloquent\Model;
6
7  class Job extends Model
8  {
9      //
10 }
11
```



```
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Job extends Model
9 {
10     /** @use HasFactory<\Database\Factories\JobFactory> */
11     use HasFactory;
12
13     protected $table = 'job_listings';
14 }
15
```

If the table name is not the same as the model name, then I have to define it using a protected variable.

```
File Edit Selection View Go Run ... workopia

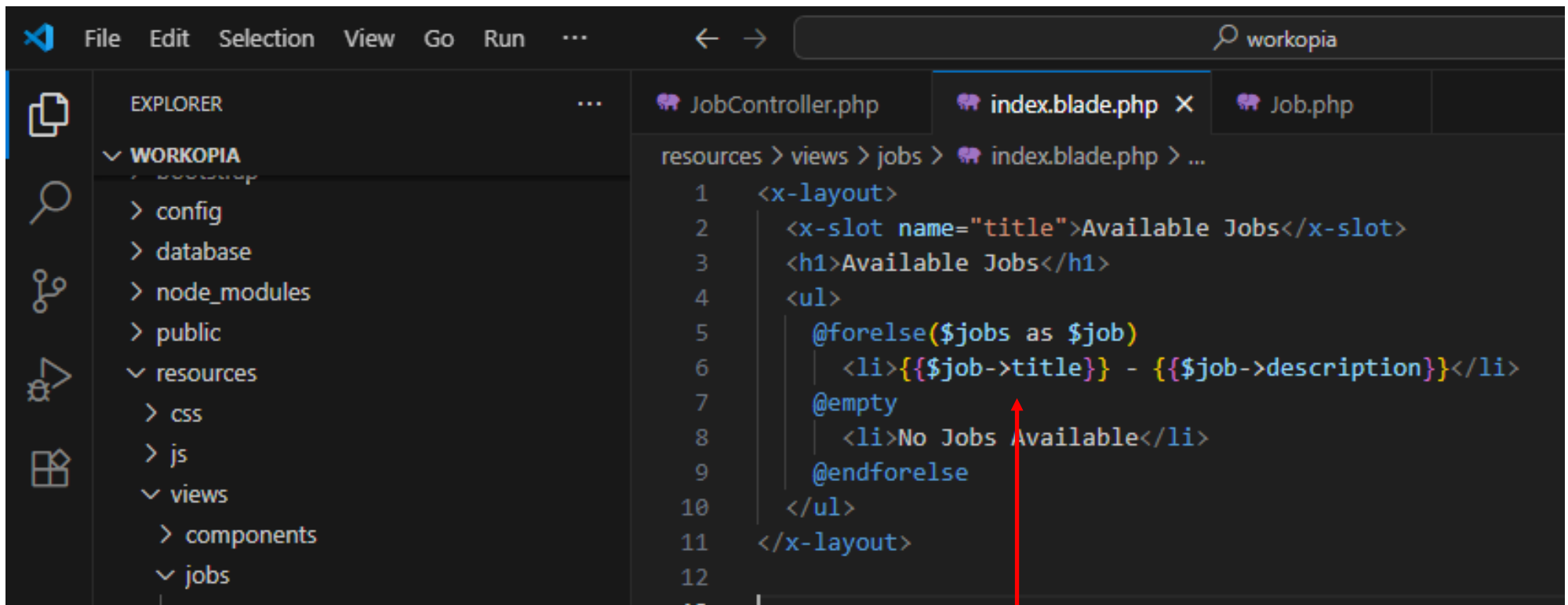
EXPLORER
WORKOPIA
  app
    Http\Controllers
      Controller.php
      HomeController.php
      JobController.php
    Models
      Job.php
      User.php
    Providers
    View
    bootstrap
    config
    database
    node_modules
    public

JobController.php X Job.php
app > Http > Controllers > JobController.php > JobController > index
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use Illuminate\View\View;
7 use App\Models\Job;
8
9 class JobController extends Controller
10 {
11     public function index(): View
12     {
13         $jobs = Job::All();
14         return view('jobs.index', compact('jobs'));
15     }
16
17     public function create(): View
```

In the Jobs controller, I now need to bring in the App\Model\Job

I can now use an Eloquent method called all from App\Model\Job to get all jobs in the database. Note that I can also use the with syntax to return the view which looks a little cleaner than the compact syntax.

```
return view('jobs.index')->with('jobs', $jobs);
```

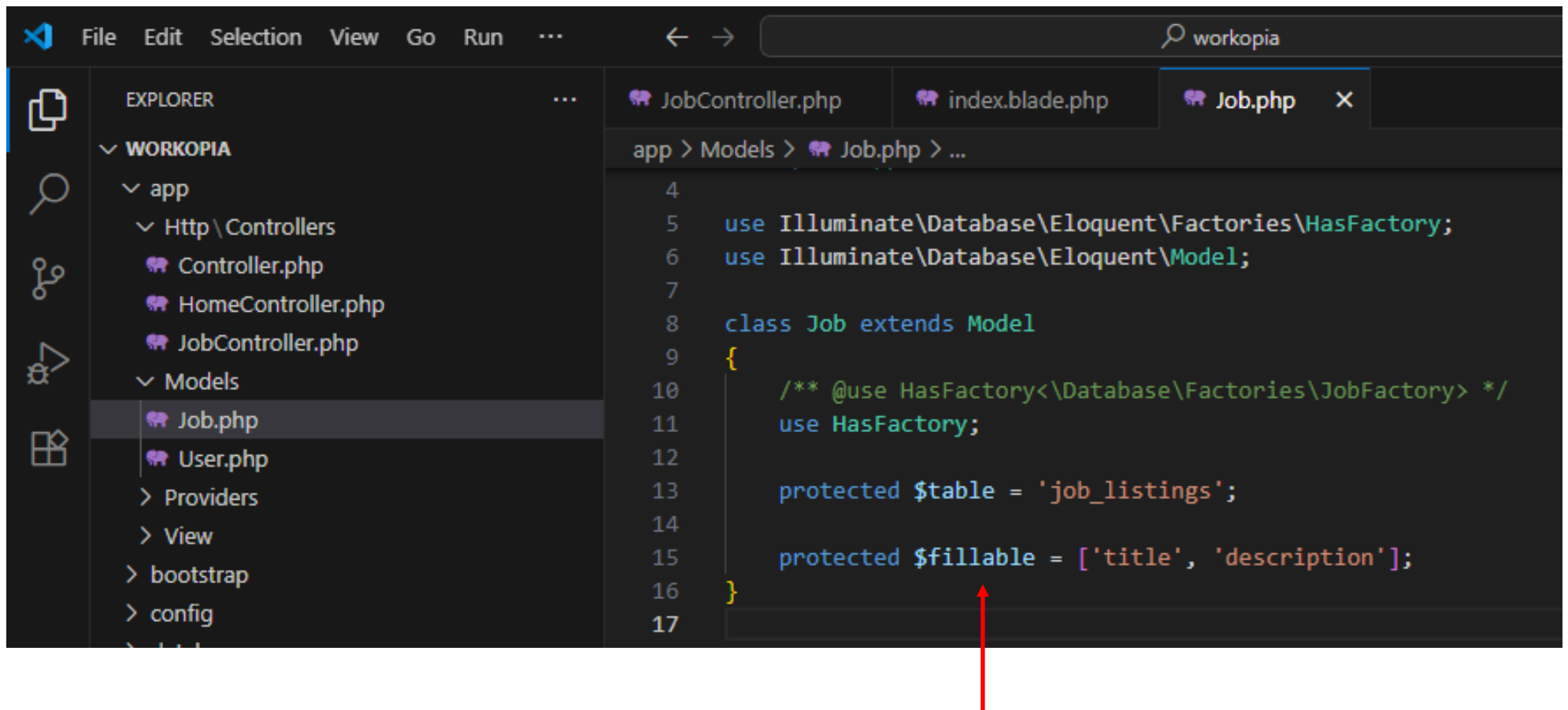


Also in the view, I am going to change the way we access the data from:
array syntax - `{{ $jobs['title'] }}`
to arrow syntax - `{{ $jobs->title }}`
This is a cleaner more modern way of writing code.

Available Jobs

No Jobs Available

Now when I reload the 'jobs' page, it shows no jobs because the database job_listings table is currently empty.



```
4
5 use Illuminate\Database\Eloquent\Factories\HasFactory;
6 use Illuminate\Database\Eloquent\Model;
7
8 class Job extends Model
9 {
10     /** @use HasFactory<\Database\Factories\JobFactory> */
11     use HasFactory;
12
13     protected $table = 'job_listings';
14
15     protected $fillable = ['title', 'description'];
16 }
17
```

Additionally, in the model, by default, Laravel has protections to stop users from changing data in fields of the database. Since in the next lesson, I am going to be using tinker, I need to enable mass assignment by defining a property called fillable with the columns that I want to add. Without this tinker would throw an error when trying to add data.

6.3 Tinker & CRUD Operations

```
PS C:\Users\ellio\Herd\workopia> php artisan tinker
Psy Shell v0.12.8 (PHP 8.4.8 - cli) by Justin Hileman
> App\Models\Job::All()
= Illuminate\Database\Eloquent\Collection {#2288
    all: [],
}

> Schema::getColumnListing('job_listings')
= [
    "id",
    "title",
    "description",
    "created_at",
    "updated_at",
]

>> $job = App\Models\Job::Class
= "App\Models\Job"

> $job::All()
= Illuminate\Database\Eloquent\Collection {#6194
    all: [],
}
```

Ticker is a tool that I can interact with the database using Laravel language rather than pure SQL. For example, App\Models\Job::All() will return me all rows of data in the table. In this case an empty array. Schema::getColumnListing('job_listing') will return an array of the column names.

Ticker is a tool that I can interact I can assign a Laravel method to a variable in tinker and use it in shorthand to get data \$job::All(). Again, it is returning empty array.

```
$job::create(['title' => 'Job One', 'description' => 'This is a description for job one'])
= App\Models\Job {#6255
  title: "Job One",
  description: "This is a description for job one",
  updated_at: "2025-06-22 13:10:06",
  created_at: "2025-06-22 13:10:06",
  id: 1,
}

>> $job::create(['title' => 'Job Two', 'description' => 'This is a description for job two'])
= App\Models\Job {#6196
  title: "Job Two",
  description: "This is a description for job two",
  updated_at: "2025-06-22 13:12:01",
  created_at: "2025-06-22 13:12:01",
  id: 2,
}

> $job::create(['title' => 'Job Three', 'description' => 'This is a description for job
three'])
= App\Models\Job {#6216
  title: "Job Three",
  description: "This is a description for job three",
  updated_at: "2025-06-22 13:12:16",
  created_at: "2025-06-22 13:12:16",
  id: 3,
}

>
```

**I can use Tinker to insert rows
into the database, using Laravel
pre-defined methods.**

RIP SQL?

```

> $job::find(1)
= App\Models\Job {#6254
    id: 1,
    title: "Job One",
    description: "This is an description for job one",
    created_at: "2025-06-22 13:10:06",
    updated_at: "2025-06-22 13:10:06",
}

>> $job::find(1)->update(['title' => 'Updated Job One'])
= true

>

```

I can use the Laravel method find() passing in the id value.

I can string methods together. For example, I can use find() then I can update column values.

| id
[PK] bigint | title
character varying (255) | description
text | created_at
timestamp without time zone | updated_at
timestamp without time zone |
|-------------------|----------------------------------|-------------------------------------|---|---|
| 1 | Updated Job One | This is an description for job one | 2025-06-22 13:10:06 | 2025-06-22 13:17:01 |
| 2 | Job Two | This is a description for job two | 2025-06-22 13:12:01 | 2025-06-22 13:12:01 |
| 3 | Job Three | This is a description for job three | 2025-06-22 13:12:16 | 2025-06-22 13:12:16 |

From tinker, I see it returned a boolean true and if I check in the database, I can see that the update CRUD operation has been performed.

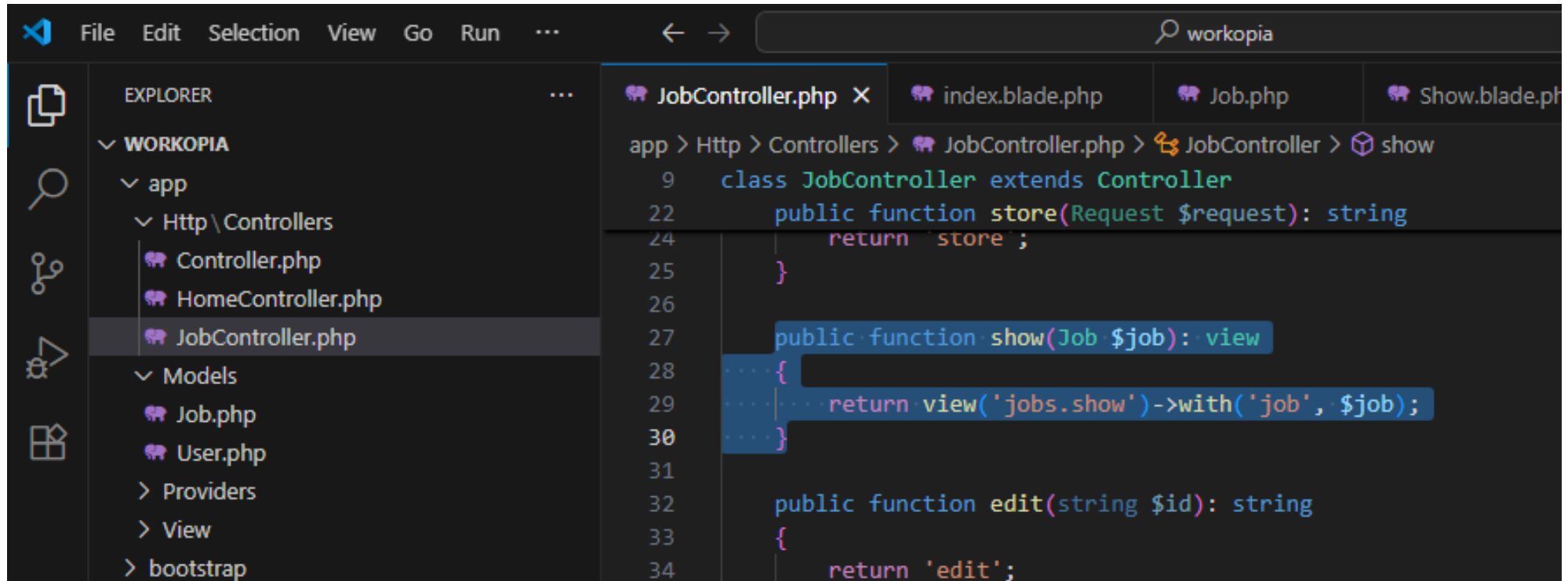
```
> $job::find(3)->delete(3)  
= true
```

I can chain Find and delete

```
>
```

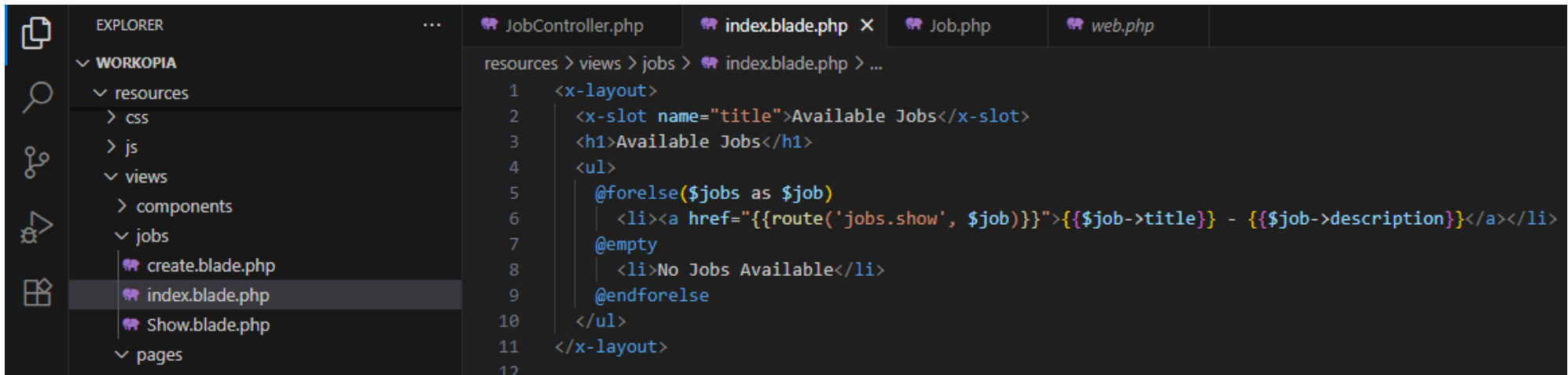
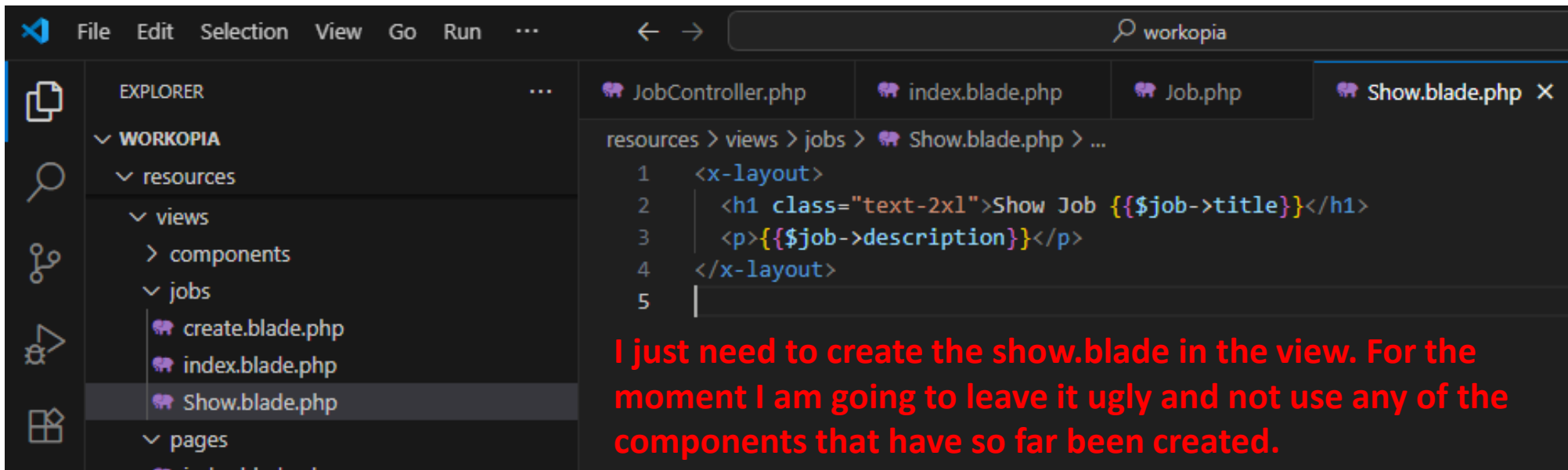
| id
[PK] bigint | title
character varying (255) | description
text | created_at
timestamp without time zone | updated_at
timestamp without time zone |
|-------------------|----------------------------------|------------------------------------|---|---|
| 1 | Updated Job One | This is an description for job one | 2025-06-22 13:10:06 | 2025-06-22 13:17:01 |
| 2 | Job Two | This is a description for job two | 2025-06-22 13:12:01 | 2025-06-22 13:12:01 |

6.4 Model Binding & Single Job Listing



```
public function show(Job $job): view
{
    return view('jobs.show')->with('job', $job);
}
```

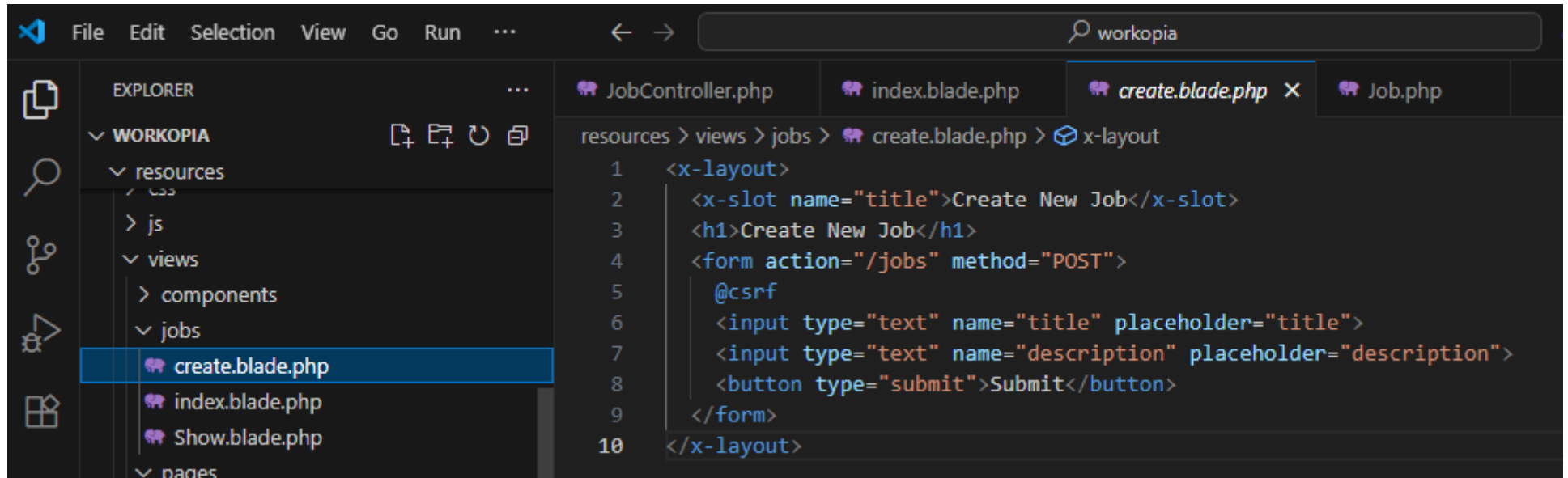
I can bind the model to the view in the controller.



`<li><a href="{{route('jobs.show', $job)}}">{{$job->title}}</a> - {{$job->description}}</li>`

In the index.blade view, I can make the list item a clickable link that takes me to the jobs/show view by using the route helper and passing in the \$job on the a href element.

6.5 Create Job Listing



The screenshot shows a code editor with the following structure:

- Explorer: WORKOPIA > resources > views > jobs > create.blade.php
- File Tabs: JobController.php, index.blade.php, create.blade.php (active), Job.php
- Code Editor: resources > views > jobs > create.blade.php > x-layout

```
1 <x-layout>
2   <x-slot name="title">Create New Job</x-slot>
3   <h1>Create New Job</h1>
4   <form action="/jobs" method="POST">
5     @csrf
6     <input type="text" name="title" placeholder="title">
7     <input type="text" name="description" placeholder="description">
8     <button type="submit">Submit</button>
9   </form>
10 </x-layout>
```

The create.blade just has a simple form in it that passes via POST the two values to jobs controller

```
<?php
```

```
namespace App\Http\Controllers;
```

```
use Illuminate\Http\Request;
```

```
use Illuminate\View\View;
```

```
use App\Models\Job;
```

```
use Illuminate\Http\RedirectResponse;
```

```
class JobController extends Controller
```

```
...
```

```
public function store(Request $request): RedirectResponse
```

```
{
```

```
    $title = $request->input('title');
```

```
    $description = $request->input('description');
```

```
    Job::create([  
        'title' => $title,  
        'description' => $description  
    ]);
```

```
    return redirect()->route('jobs.index');
```

```
}
```

An insert operation goes to the store method in the jobs controller where I can extract the POST values from the httpRequest object.

I can use the create method like previously seen in php artisan tinker to perform the SQL INSERT operation

Once the insert is done, I want to return a redirect to the jobs page so I can bring in the Laravel Http\RedirectResponse class and use the route helper to take me back to the jobs.index page.

6.6 Input Validation & Errors

```
public function store(Request $request): RedirectResponse
{
    $validatedData = $request->validate([
        'title' => 'required|string|max:255',
        'description' => 'required|string'
    ]);

    Job::create([
        'title' => $validatedData['title'],
        'description' => $validatedData['description']
    ]);

    return redirect()->route('jobs.index');
}
```

I am going to modify the store method in the job controller. First, I define an array using the Laravel validate method on the request object. Within the array I define the desired validation criteria to the fields.

I can then pass in the \$validatedData into the create method.

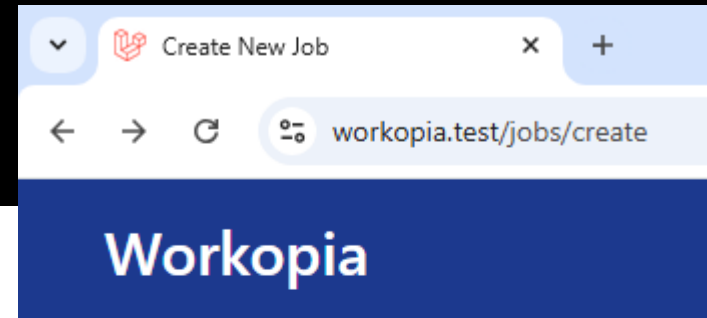
```

<x-layout>
  <x-slot name="title">Create New Job</x-slot>
  <h1>Create New Job</h1>
  <form action="/jobs" method="POST">
    @csrf
    <input type="text" name="title" placeholder="title">
    @error('title')
      <div class="text-red-500 mt-2 text-sm">{{ $message }}</div>
    @enderror
    <input type="text" name="description" placeholder="description">
    @error('description')
      <div class="text-red-500 mt-2 text-sm">{{ $message }}</div>
    @enderror
    <button type="submit">Submit</button>
  </form>
</x-layout>

```

In the create.blade, I have access to a **\$message** variable that I can use in an **@error** directive

If I now test this, I can see that in the form, if the fields are empty and the submit button is clicked, then the relevant error message is displayed, based on the constraints defined in the validate method.



Create New Job

title

The title field is required.

description

The description field is required.

Submit

6.7 Job Schema update Migration

```
PS C:\Users\ellio\Herd\workopia> php artisan make:migration  
add_fields_to_job_listings_table -table=job_listings
```

INFO Migration

```
[C:\Users\ellio\Herd\workopia\database\Migrations\2025_06_22_150737_add_fields_to_j  
ob_listings_table.php] created successfully.
```

I want to add more fields to the job_listings
table so I create a new migration

```
PS C:\Users\ellio\Herd\workopia>
```

```
use Illuminate\Database\Migrations\Migration;  
use Illuminate\Database\Schema\Blueprint;  
use Illuminate\Support\Facades\Schema;  
use illuminate\Support\Facades\DB;
```

```
return new class extends Migration  
{
```

```
    public function up(): void  
    {
```

```
        // delete all table rows (truncate)  
        DB::table('job_listings')->truncate();
```

```
        Schema::table('job_listings', function (Blueprint $table) {
```

In the migration file, I first want to
truncate all rows in the jobs table
to remove the test data that I
have been working with so far.

```
// Modify the table schema
Schema::table('job_listings', function (Blueprint $table) {
    $table->integer('salary');
    $table->string('tags')->nullable();
    $table->enum('job_type', ['Full-Time', 'Part-Time', 'Contract', 'Temporary',
'Internship', 'Volunteer', 'On-Call'])->default('Full-Time');
    $table->boolean('remote')->default(false);
    $table->text('requirements')->nullable();
    $table->text('benefits')->nullable();
    $table->string('address')->nullable();
    $table->string('city');
    $table->string('state');
    $table->string('zipcode')->nullable();
    $table->string('contact_email');
    $table->string('contact_phone')->nullable();
    $table->string('company_name');
    $table->text('company_description')->nullable();
    $table->string('company_logo')->nullable();
    $table->string('company_website')->nullable();
});
```

In the up, I want to add some more fields to the table so I define them.

```
public function down(): void
{
    Schema::table('job_listings', function (Blueprint $table) {
        $table->dropColumn(['salary', 'tags', 'job_type', 'remote', 'requirements',
        'benefits', 'address', 'city', 'state', 'zipcode', 'contact_email',
        'contact_phone', 'company_name', 'company_description', 'company_logo',
        'company_website']);
    });
}
```

The down is to undo the changes made in the UP so I just want to drop the added columns.



EXPLORER

WORKOPIA

app

Http \ Controllers

Controller.php
HomeController.php
JobController.php

Models

Job.php
User.php

> Providers

> View

> bootstrap

> config

database

> factories

migrations

0001_01_01_000000_create_users_table....
0001_01_01_000001_create_cache_table....
0001_01_01_000002_create_jobs_table.p...
2025_06_21_213659_create_job_listings_...
2025_06_22_150737_add_fields_to_job_li...

> seeders

> OUTLINE



JobController.php

index.blade.php

create.blade.p

app > Models > Job.php > Job > \$fillable

8 class Job extends Model

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

protected \$table = 'job_listings';

protected \$fillable = [

'title',

'description',

'salary',

'tags',

'job_type',

'remote',

'requirements',

'benefits',

'address',

'city',

'state',

'zipcode',

'contact_email',

'contact_phone',

'company_name',

'company_description',

'company_logo',

'company_website'

];

}

In the job model
all these
additional fields
need to be
added into the
\$fillable because
data to this table
is going to be
added via a
form.

6.8 Eloquent Relationships

Types Of Relationships

✓ One-To-One

Relationship between two tables where each record in one table is related to exactly one record in the other table.

Example: User profiles

✓ One-To-Many

Relationship between two tables where each record in one table is related to zero, one, or many records in the other table.

Example: User blog posts

✓ Many-To-Many

Relationship between two tables where each record in one table is related to zero, one, or many records in the other table, and vice versa.

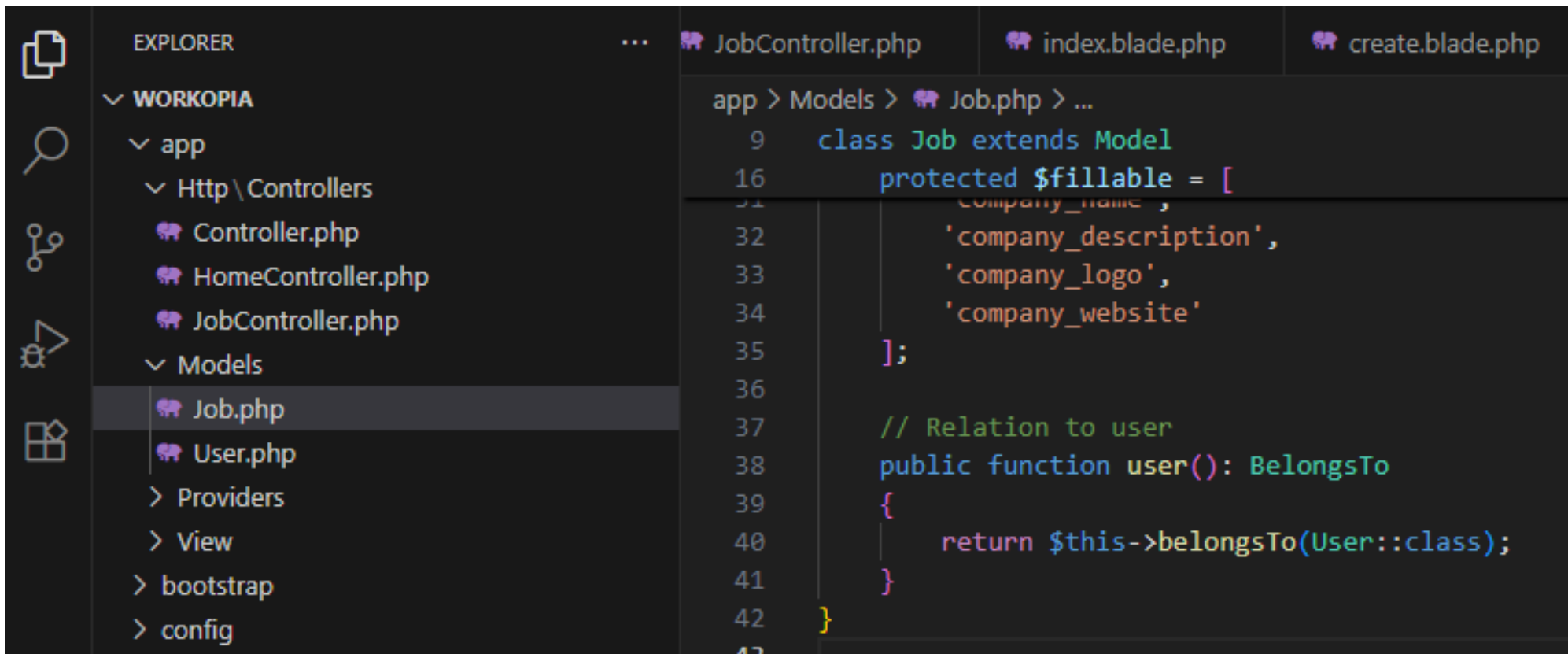
Example: Student courses

I want to define a **one-2-many** relation ship between a user and a job_listing. i.e. a user can post more than one job.

Code Example

```
$user = User::find(1);  
$jobListings = $user->jobListings;
```

```
$jobListing = JobListing::find(1);  
$user = $jobListing->user;
```



```
EXPLORER
WORKOPIA
  app
    Http \ Controllers
      Controller.php
      HomeController.php
      JobController.php
    Models
      Job.php
      User.php
    Providers
    View
    bootstrap
    config

JobController.php
index.blade.php
create.blade.php

app > Models > Job.php > ...
9      class Job extends Model
16         protected $fillable = [
31             'company_name',
32             'company_description',
33             'company_logo',
34             'company_website'
35         ];
36
37         // Relation to user
38         public function user(): BelongsTo
39         {
40             return $this->belongsTo(User::class);
41         }
42     }
```

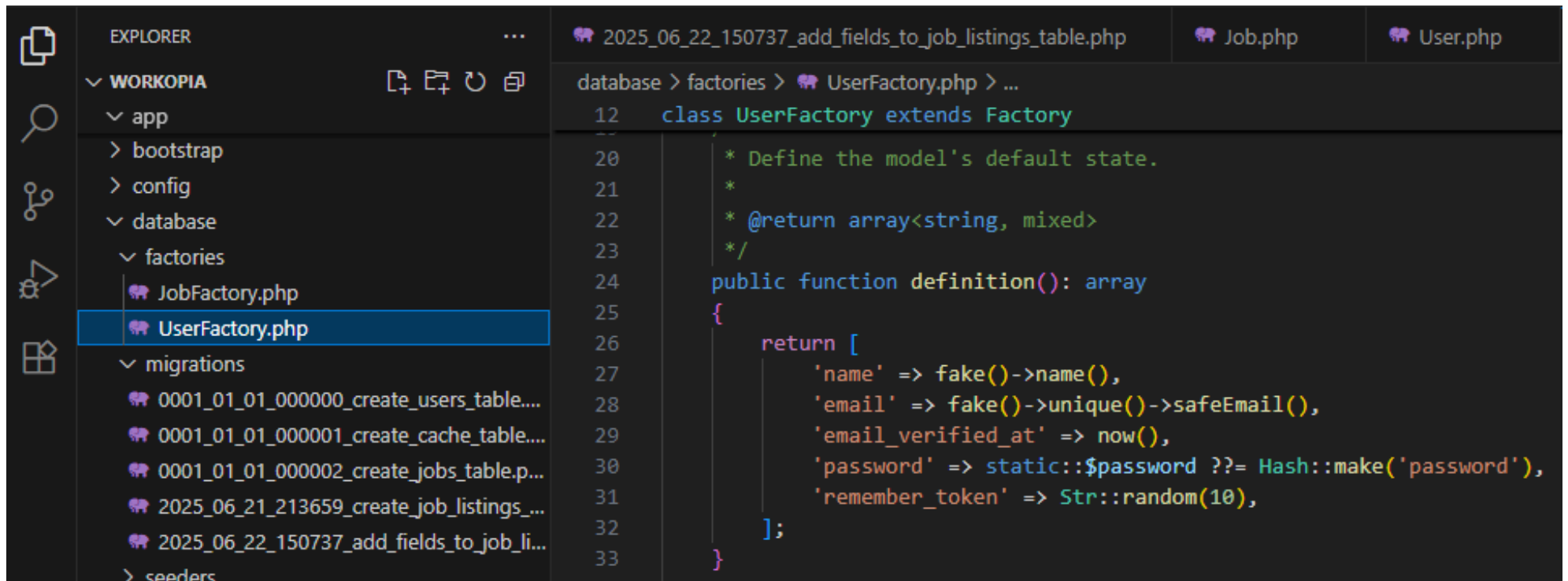
In the jobs model I want to define the relationship between job and user. A job belongs to a user. **\$this** job belongsTo the **user::class**.

6.9 Using Factories

What Are Factories?

Factories are a way to define the attributes of a model in a reusable way. They allow you to define a blueprint for creating model instances with predefined attributes. This is useful when you need to seed your database with sample data or when you need to create model instances in your tests.

Laravel provides a factory class that you can use to define factories for your models. Factories are typically stored in the ``database/factories`` directory of your Laravel application.



The screenshot shows a code editor with a dark theme. On the left is the 'EXPLORER' sidebar showing a file tree. The tree is expanded to 'database > factories', where 'UserFactory.php' is selected. The main editor area shows the contents of 'UserFactory.php'. The code defines a 'UserFactory' class that extends 'Factory'. It includes a docblock comment for the 'definition()' method, which returns an array of user attributes. The attributes include a fake name, a unique fake email, a current timestamp for 'email_verified_at', a hashed password, and a random 10-character string for 'remember_token'.

```
12 class UserFactory extends Factory
13 {
14     /**
15      * Define the model's default state.
16      *
17      * @return array<string, mixed>
18      */
19     public function definition(): array
20     {
21         return [
22             'name' => fake()->name(),
23             'email' => fake()->unique()->safeEmail(),
24             'email_verified_at' => now(),
25             'password' => static::$password ??= Hash::make('password'),
26             'remember_token' => Str::random(10),
27         ];
28     }
29 }
```

Laravel comes bundled with Faker, which can be used to generate dummy sample data. In the framework there is a database/factories class which has a prebuilt class to automatically insert dummy user data for testing.

What this is doing is providing a fake name, fake email but unique within the database, a fake password which is hashed, an email address that by default is unverified and a remember token that is a random 10-character string. Note that the plain text for the password is 'password'

```
PS C:\Users\ellio\Herd\workopia> php artisan tinker
Psy Shell v0.12.8 (PHP 8.4.8 - cli) by Justin Hileman
> \App\Models\User::factory()->create();
= App\Models\User {#6269
    name: "Bethel Rowe",
    email: "imacejkovic@example.net",
    email_verified_at: "2025-06-22 17:51:45",
    #password: "$2y$12$J5K7pxEkgEXQzdsj7VQFM0hDoP/r9Ry0qjVg7D31sCeh3ZXtn.c/G",
    #remember_token: "Sql9809e2x",
    updated_at: "2025-06-22 17:51:45",
    created_at: "2025-06-22 17:51:45",
    id: 1,
}
```

Tinker & Faker can be used to insert fake dummy data into the user database table for testing purposes.

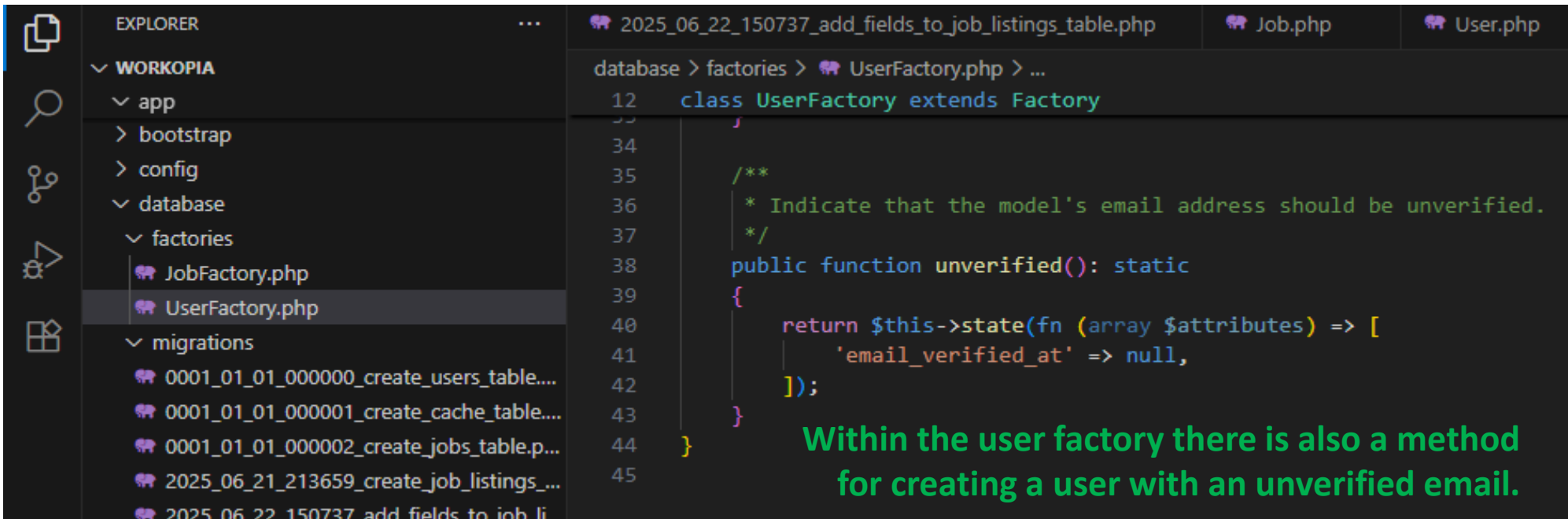
| id
[PK] bigint | name
character varying (255) | email
character varying (255) | email_verified_at
timestamp without time zone | password
character varying (255) |
|-------------------|---------------------------------|----------------------------------|--|--|
| 1 | Bethel Rowe | imacejkovic@example.net | 2025-06-22 17:51:45 | \$2y\$12\$J5K7pxEkgEXQzdsj7VQFM0hDoP/r9Ry0qjVg |

I can repeat this command multiple times to add more dummy users.

| id
[PK] bigint | name
character varying (255) | email
character varying (255) | email_verified_at
timestamp without time zone | password
character varying (255) |
|-------------------|---------------------------------|----------------------------------|--|-------------------------------------|
| 1 | Bethel Rowe | imacejkovic@example.net | 2025-06-22 17:51:45 | \$2y\$12\$J5K7pxEkgEXQzdsj7 |
| 2 | Dr. Jared Treutel Sr. | braun.rudolph@example.com | 2025-06-22 17:56:48 | \$2y\$12\$J5K7pxEkgEXQzdsj7 |
| 3 | Rebekah Schoen | alfonso.bins@example.org | 2025-06-22 17:57:06 | \$2y\$12\$J5K7pxEkgEXQzdsj7 |
| 4 | Karianne Gutmann I | gerardo.halvorson@example.com | 2025-06-22 17:57:24 | \$2y\$12\$J5K7pxEkgEXQzdsj7 |
| 5 | Ms. Petra Kuhlman Jr. | jpouros@example.org | 2025-06-22 17:57:26 | \$2y\$12\$J5K7pxEkgEXQzdsj7 |
| 6 | Prof. Brian Emard | schmitt.polly@example.net | 2025-06-22 17:57:27 | \$2y\$12\$J5K7pxEkgEXQzdsj7 |
| 7 | Cedrick Franecki | precious.smith@example.net | 2025-06-22 17:57:28 | \$2y\$12\$J5K7pxEkgEXQzdsj7 |
| 8 | Jordi Mohr | lbauch@example.org | 2025-06-22 17:57:28 | \$2y\$12\$J5K7pxEkgEXQzdsj7 |

```
PS C:\Users\ellio\Herd\workopia> php artisan tinker
Psy Shell v0.12.8 (PHP 8.4.8 - cli) by Justin Hileman
> \App\Models\User::factory()->count(10)->create();
= Illuminate\Database\Eloquent\Collection {#6192
  all: [
    App\Models\User {#6278
```

I can also use `count()` to pass in as many rows as I want to the table. In this case I am adding another 10 fake users.



```
2025_06_22_150737_add_fields_to_job_listings_table.php Job.php User.php
database > factories > UserFactory.php > ...
12 class UserFactory extends Factory
33 {
34     /**
35      *
36      * Indicate that the model's email address should be unverified.
37      */
38     public function unverified(): static
39     {
40         return $this->state(fn (array $attributes) => [
41             'email_verified_at' => null,
42         ]);
43     }
44 }
45
```

Within the user factory there is also a method for creating a user with an unverified email.

```
PS C:\Users\ellio\Herd\workopia> php artisan tinker
Psy Shell v0.12.8 (PHP 8.4.8 - cli) by Justin Hileman
> \App\Models\User::factory()->unverified()->create();
= App\Models\User {#6248
    name: "Arthur Russel Jr.",
    email: "hubert.metz@example.org",
    email_verified_at: null,
    #password: "$2y$12$M6SLV1dqW2B207Iw6tNSZu10fxNYzK38KL7OLmqUYFG3tV9B7A982",
    #remember_token: "xaEGjtwQ08",
    updated_at: "2025-06-22 18:13:08",
    created_at: "2025-06-22 18:13:08",
    id: 19,
}
```

I can use the unverified method in tinker to insert a row into the database that has email_verified set to NULL

6.10 Creating Factories & Faker Library

```
PS C:\Users\ellio\Herd\workopia> php artisan make:factory JobFactory
```

```
INFO Factory [C:\Users\ellio\Herd\workopia\database\factories\JobFactory.php]  
created successfully.
```

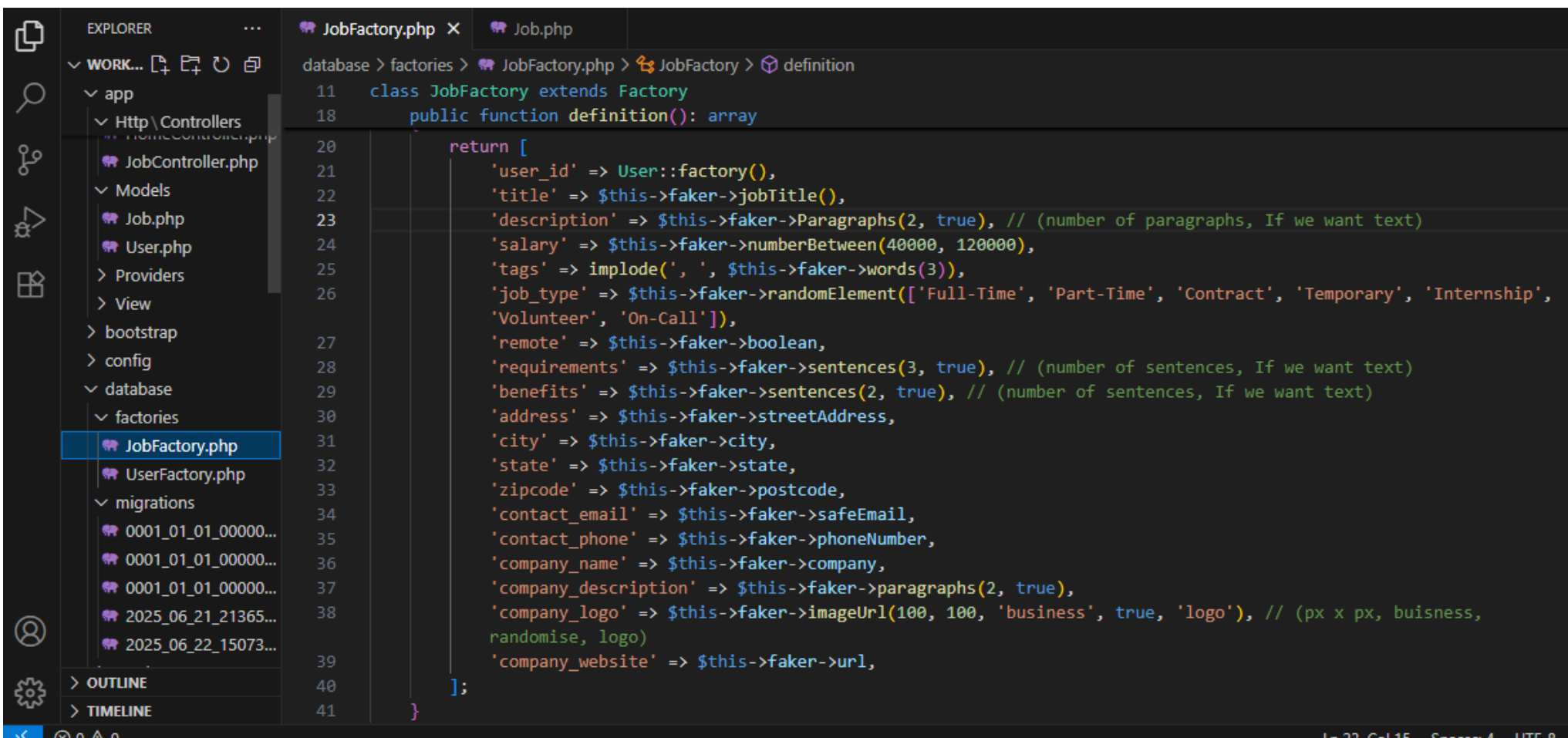
```
PS C:\Users\ellio\Herd\workopia>
```

Create a new factory for the Job_listing table

The complete list of possible faker fields can be found on the website. It varies from country to country.

<https://faker.readthedocs.io/en/master/index.html>

<https://faker.readthedocs.io/en/master/locales.html>



```
database > factories > JobFactory.php > JobFactory > definition
11 class JobFactory extends Factory
18     public function definition(): array
20     {
21         return [
22             'user_id' => User::factory(),
23             'title' => $this->faker->jobTitle(),
24             'description' => $this->faker->Paragraphs(2, true), // (number of paragraphs, If we want text)
25             'salary' => $this->faker->numberBetween(40000, 120000),
26             'tags' => implode(', ', $this->faker->words(3)),
27             'job_type' => $this->faker->randomElement(['Full-Time', 'Part-Time', 'Contract', 'Temporary', 'Internship',
28             'Volunteer', 'On-Call']),
29             'remote' => $this->faker->boolean,
30             'requirements' => $this->faker->sentences(3, true), // (number of sentences, If we want text)
31             'benefits' => $this->faker->sentences(2, true), // (number of sentences, If we want text)
32             'address' => $this->faker->streetAddress,
33             'city' => $this->faker->city,
34             'state' => $this->faker->state,
35             'zipcode' => $this->faker->postcode,
36             'contact_email' => $this->faker->safeEmail,
37             'contact_phone' => $this->faker->phoneNumber,
38             'company_name' => $this->faker->company,
39             'company_description' => $this->faker->paragraphs(2, true),
40             'company_logo' => $this->faker->imageUrl(100, 100, 'business', true, 'logo'), // (px x px, buisness,
41             'company_website' => $this->faker->url,
42         ];
43     }
```

Within 'Factories\JobFactories' I can define an array if dummy data and what kind of faker values I want to use.


```
PS C:\Users\ellio\Herd\workopia> php artisan tinker
Psy Shell v0.12.8 (PHP 8.4.8 - cli) by Justin Hileman
```

```
> \App\Models\Job::factory()->create();
```

```
= App\Models\Job {#6255
```

```
    user_id: 21,
    title: "Photographic Developer",
    description: ""
```

```
    Sunt dicta nesciunt sit enim iure ipsum. Aut tenetur fugit deleniti sed maxime magnam.
    Non illum nam aperiam.\n
```

```
    \n
```

```
    Libero maiores quo velit quia occaecati. Rerum recusandae sed minus perspiciatis.
    Expedita nostrum harum temporibus expedita. Magni repellat delectus iste unde. Ut corrupti
    autem mollitia.
```

```
    "",
```

```
    salary: 75053,
    tags: "quod, dolorum, voluptatem",
    job_type: "Volunteer",
    remote: false,
```

```
    requirements: "Minima dolorum aut debitis consequatur dolorem. Aut qui ex quos
    consequatur. Consequatur ad maxime modi expedita.",
```

```
    benefits: "Optio perferendis ut nemo omnis. Ea distinctio suscipit quia.",
```

```
    address: "1443 Conn Hills",
```

```
    city: "Port Kylerberg",
```

```
    state: "Georgia",
```

```
    zipcode: "06495-3138",
```

```
    contact_email: "frida23@example.com",
```

```
    contact_phone: "1-925-421-2461",
```

```
    company_name: "Huel-Baumbach",
```

```
...
```

I can use the factory to insert
a row of dummy data.


```
...
  company_description: ""
    Est et est assumenda dolore facilis voluptas. Neque ipsum et odio dolorem alias. Magni
assumenda officia distinctio commodi quia. Veniam est aut fugiat non est reiciendis.\n
    \n
    Aperiam quia dolorem neque ad inventore. Est officia velit maxime itaque ipsam quia
repellendus. Hic sed corporis maiores nesciunt. Soluta consequatur quibusdam vel dolorum sunt.
    "",
  company_logo: "https://via.placeholder.com/100x100.png/003333?text=business+logo+sed",
  company_website: "http://www.grady.com/",
  updated_at: "2025-06-22 19:12:22",
  created_at: "2025-06-22 19:12:22",
  id: 1,
}

>

> \App\Models\Job::factory()->count(5)->create();
// I can add 5 more rows

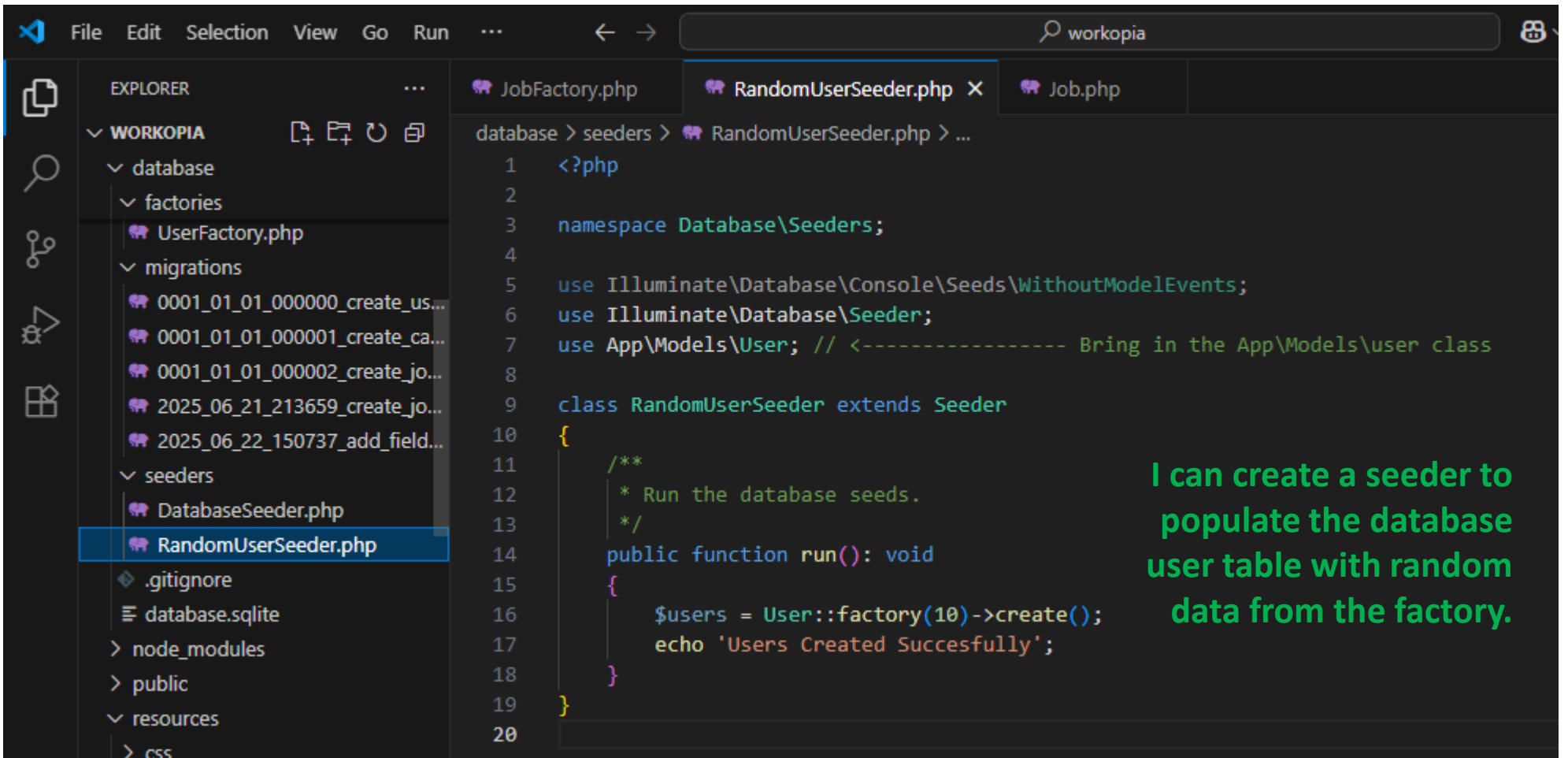
> \App\Models\Job::all();
// to view all rows
```

6.11 Creating Seeders

```
PS C:\Users\ellio\Herd\workopia> php artisan make:seeder RandomUserSeeder
```

```
INFO Seeder [C:\Users\ellio\Herd\workopia\database\seeders\RandomUserSeeder.php] created successfully.
```

```
PS C:\Users\ellio\Herd\workopia>
```



The screenshot shows the Visual Studio Code interface. The Explorer sidebar on the left displays the project structure, with the `RandomUserSeeder.php` file highlighted under the `seeders` directory. The main editor area shows the content of `RandomUserSeeder.php`, which includes a namespace declaration, use statements for Laravel's seeding classes, and a `run()` method that creates 10 random users.

```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Console\Seeds\WithoutModelEvents;
6 use Illuminate\Database\Seeder;
7 use App\Models\User; // <----- Bring in the App\Models\user class
8
9 class RandomUserSeeder extends Seeder
10 {
11     /**
12      * Run the database seeds.
13      */
14     public function run(): void
15     {
16         $users = User::factory(10)->create();
17         echo 'Users Created Successfully';
18     }
19 }
20
```

I can create a seeder to populate the database user table with random data from the factory.

```
PS C:\Users\ellio\Herd\workopia> php artisan db:seed --class=RandomUserSeeder
```

```
INFO Seeding database.
```

```
Users Created Successfully
```

```
PS C:\Users\ellio\Herd\workopia>
```

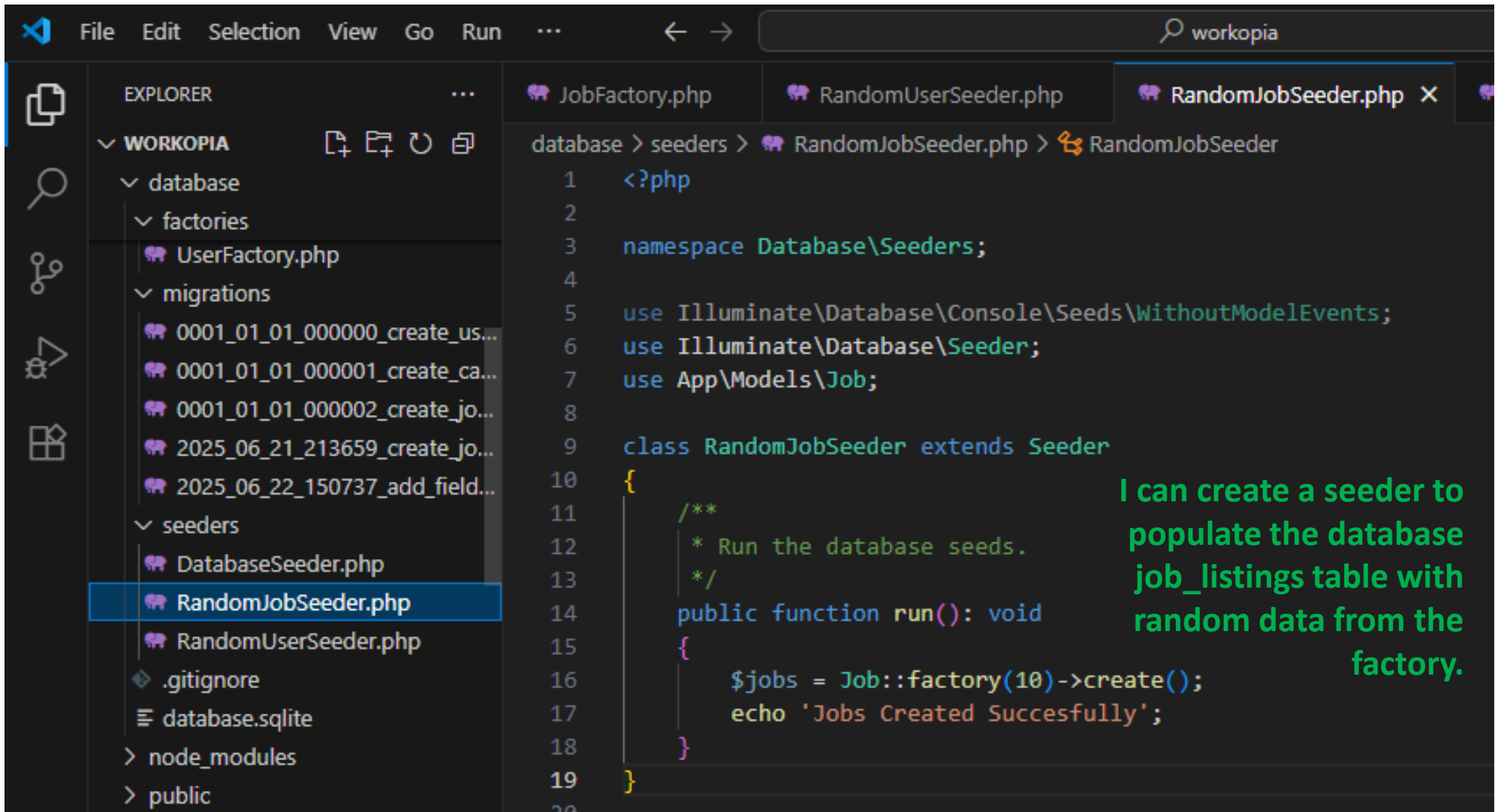
I can trigger the seeder to populate the table. Note how I provide additional parameters of class=RandomUserSeeder because otherwise it would call the Database seeder

```
PS C:\Users\ellio\Herd\workopia> php artisan make:seeder RandomJobSeeder
```

```
INFO Seeder [C:\Users\ellio\Herd\workopia\database\seeders\RandomJobSeeder.php] created successfully.
```

```
PS C:\Users\ellio\Herd\workopia>
```

Create a seeder for the Job class



The screenshot shows the Visual Studio Code interface. The Explorer view on the left displays the project structure: `WORKOPIA` > `database` > `factories` > `UserFactory.php` > `migrations` > `0001_01_01_000000_create_us...` > `0001_01_01_000001_create_ca...` > `0001_01_01_000002_create_jo...` > `2025_06_21_213659_create_jo...` > `2025_06_22_150737_add_field...` > `seeders` > `DatabaseSeeder.php` > **`RandomJobSeeder.php`** > `RandomUserSeeder.php` > `.gitignore` > `database.sqlite` > `node_modules` > `public`.

The Editor view shows the code for `RandomJobSeeder.php`:

```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Console\Seeds\WithoutModelEvents;
6 use Illuminate\Database\Seeder;
7 use App\Models\Job;
8
9 class RandomJobSeeder extends Seeder
10 {
11     /**
12      * Run the database seeds.
13      */
14     public function run(): void
15     {
16         $jobs = Job::factory(10)->create();
17         echo 'Jobs Created Successfully';
18     }
19 }
```

I can create a seeder to populate the database job_listings table with random data from the factory.

```
PS C:\Users\ellio\Herd\workopia> php artisan db:seed --class=RandomJobSeeder
```

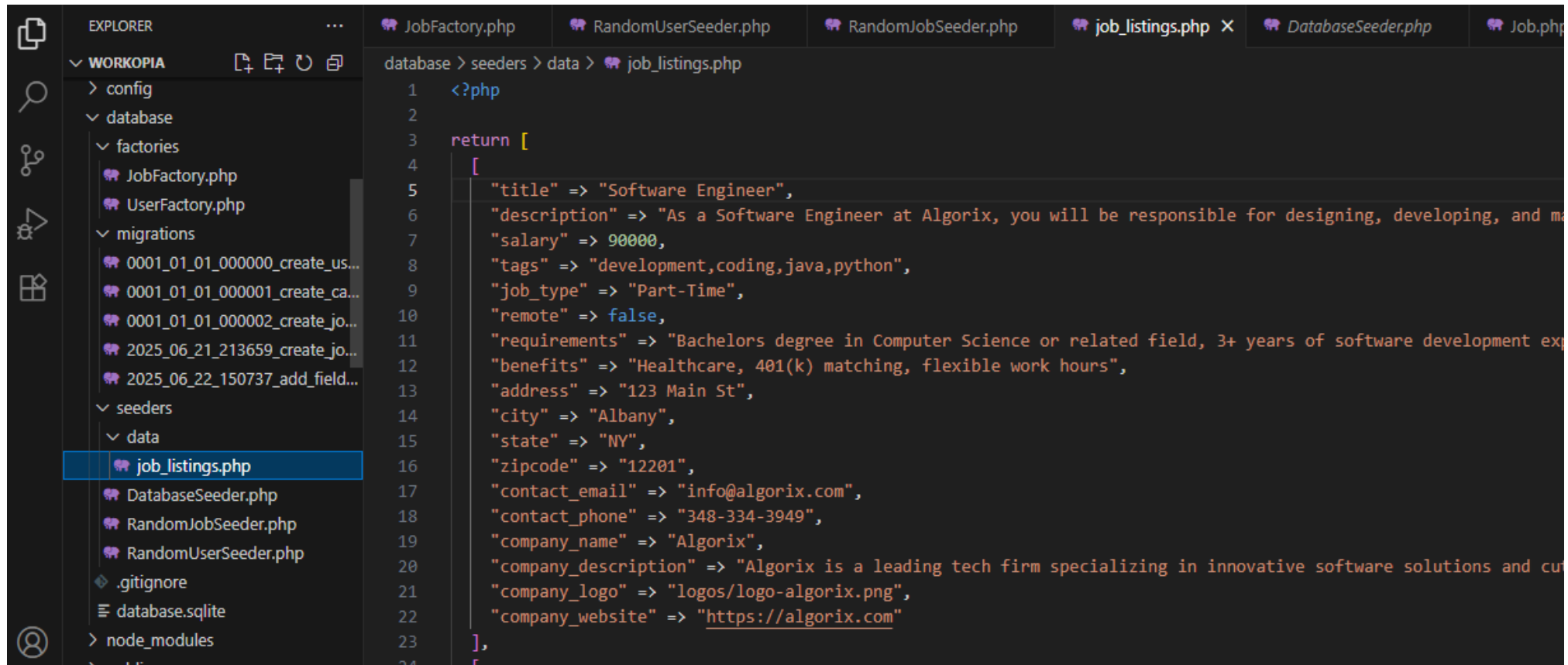
```
INFO Seeding database.
```

```
Jobs Created Successfully
```

```
PS C:\Users\ellio\Herd\workopia>
```

Triger the seeder to populate the jobs_listings table with 10 more jobs

6.12 Final Database Seeder



```
1 <?php
2
3 return [
4     [
5         "title" => "Software Engineer",
6         "description" => "As a Software Engineer at Algorix, you will be responsible for designing, developing, and ma
7         "salary" => 90000,
8         "tags" => "development,coding,java,python",
9         "job_type" => "Part-Time",
10        "remote" => false,
11        "requirements" => "Bachelors degree in Computer Science or related field, 3+ years of software development exp
12        "benefits" => "Healthcare, 401(k) matching, flexible work hours",
13        "address" => "123 Main St",
14        "city" => "Albany",
15        "state" => "NY",
16        "zipcode" => "12201",
17        "contact_email" => "info@algorix.com",
18        "contact_phone" => "348-334-3949",
19        "company_name" => "Algorix",
20        "company_description" => "Algorix is a leading tech firm specializing in innovative software solutions and cu
21        "company_logo" => "logos/logo-algorix.png",
22        "company_website" => "https://algorix.com"
23    ],
24    [
```

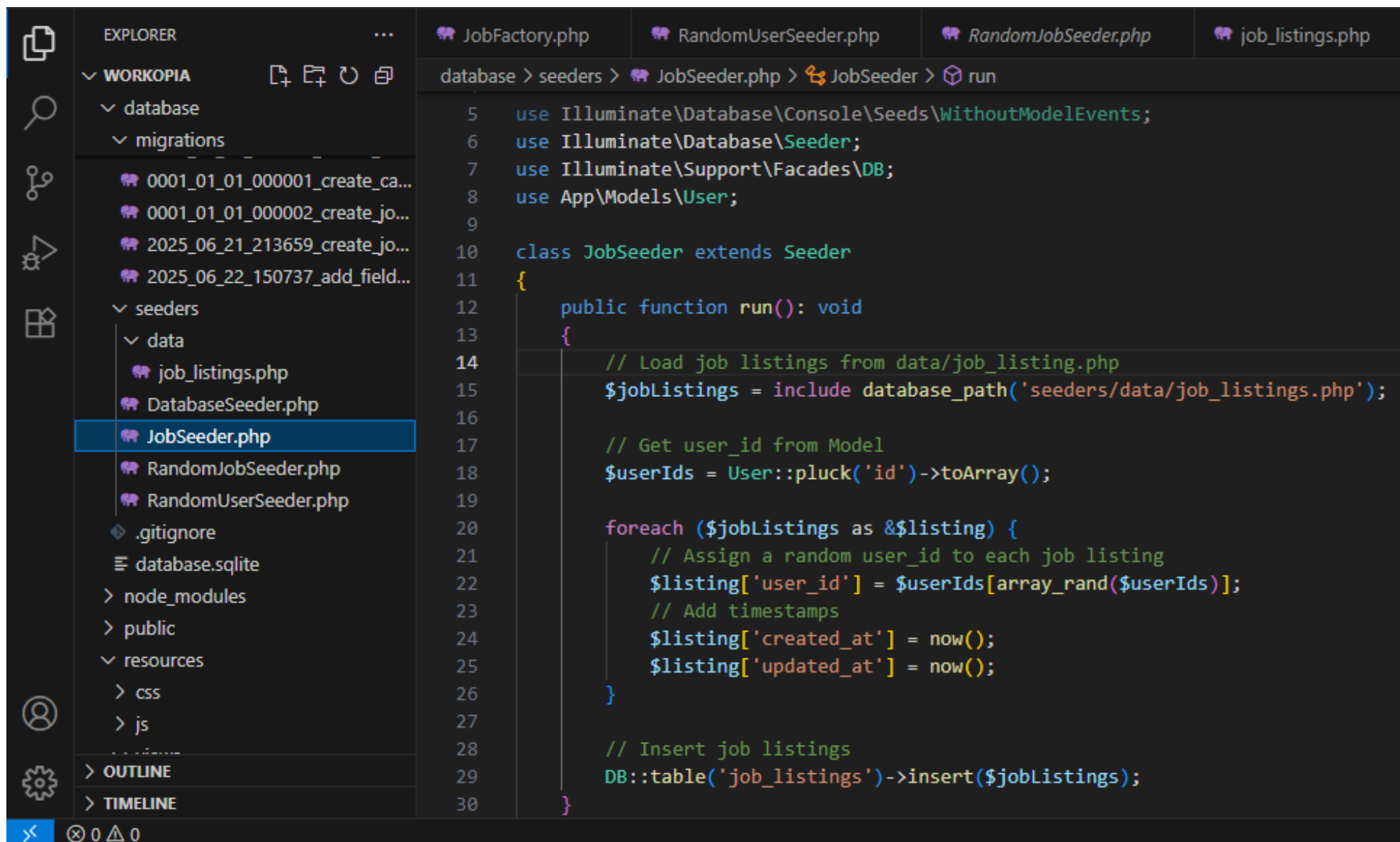
In The Seeders directory, I have created a folder called data which contains a php file that is an associative array of realistic jobs rather than the Loren Ipsum data generated by Faker.

```
PS C:\Users\ellio\Herd\workopia> php artisan make:seeder JobSeeder
```

```
INFO Seeder [C:\Users\ellio\Herd\workopia\database\seeders\JobSeeder.php] created successfully.
```

```
PS C:\Users\ellio\Herd\workopia>
```

The JobSeeder is importing the array of realistic jobs and then getting the users from the database. I iterate over the users and assign a random user to each 'job_listings' in the array and generate timestamps. Finally, the rows of job listings are inserted into the 'job_listings' table.



The screenshot displays a code editor interface with a sidebar on the left showing the project structure. The main editor area shows the code for `JobSeeder.php`.

Explorer (Left Sidebar):

- WORKOPIA
 - database
 - migrations
 - 0001_01_01_000001_create_ca...
 - 0001_01_01_000002_create_jo...
 - 2025_06_21_213659_create_jo...
 - 2025_06_22_150737_add_field...
 - seeders
 - data
 - job_listings.php
 - DatabaseSeeder.php
 - JobSeeder.php** (selected)
 - RandomJobSeeder.php
 - RandomUserSeeder.php
 - .gitignore
 - database.sqlite
 - node_modules
 - public
 - resources
 - css
 - js
 - OUTLINE
 - TIMELINE

Main Editor (JobSeeder.php):

```
5 use Illuminate\Database\Console\Seeds\WithoutModelEvents;
6 use Illuminate\Database\Seeder;
7 use Illuminate\Support\Facades\DB;
8 use App\Models\User;
9
10 class JobSeeder extends Seeder
11 {
12     public function run(): void
13     {
14         // Load job listings from data/job_listing.php
15         $jobListings = include database_path('seeders/data/job_listings.php');
16
17         // Get user_id from Model
18         $userIds = User::pluck('id')->toArray();
19
20         foreach ($jobListings as &$listing) {
21             // Assign a random user_id to each job listing
22             $listing['user_id'] = $userIds[array_rand($userIds)];
23             // Add timestamps
24             $listing['created_at'] = now();
25             $listing['updated_at'] = now();
26         }
27
28         // Insert job listings
29         DB::table('job_listings')->insert($jobListings);
30     }
31 }
```



EXPLORER



WORKOPIA

database

migrations

0001_01_01_000001_create_ca...

0001_01_01_000002_create_jo...

2025_06_21_213659_create_jo...

2025_06_22_150737_add_field...

seeders

data

job_listings.php

DatabaseSeeder.php

JobSeeder.php

RandomJobSeeder.php

RandomUserSeeder.php

.gitignore

database.sqlite

node_modules

public

resources

css

js



OUTLINE

DatabaseSeeder.php X

JobFactory.php

RandomUserSeeder.php

R

database > seeders > DatabaseSeeder.php > DatabaseSeeder > run

```
1  <?php
2
3  namespace Database\Seeders;
4
5  use App\Models\User;
6  // use Illuminate\Database\Console\Seeds\WithoutModelEvents;
7  use Illuminate\Database\Seeder;
8  use Illuminate\Support\Facades\DB;
9
10 class DatabaseSeeder extends Seeder
11 {
12     /**
13      * Seed the application's database.
14      */
15     public function run(): void
16     {
17         // Truncate tables
18         DB::table('job_listings')->truncate();
19         DB::table('users')->truncate();
20
21         $this->call(RandomUserSeeder::class);
22         $this->call(JobSeeder::class);
23     }
24 }
25
```

I have modified the database seeder to first truncate the users table and the job_listings table.

Then I call the two seeders to re-populate the two tables


```
PS C:\Users\ellio\Herd\workopia> php artisan DB:seed;
```

```
INFO Seeding database.
```

```
Database\Seeders\RandomUserSeeder ..... RUNNING
```

```
INFO Seeding database.
```

```
Database\Seeders\RandomUserSeeder ..... RUNNING
```

```
INFO Seeding database.
```

```
Database\Seeders\RandomUserSeeder ..... RUNNING
```

```
Users Created Succesfully 364 ms DONE
```

```
Database\Seeders\JobSeeder ..... RUNNING 364 ms DONE
```

Now I trigger the database seeder rather than a particular seeder

```
PS C:\Users\ellio\Herd\workopia> file and it performs all the table truncation and row insertions
```

| id
[PK] bigint | created_at
timestamp without time zone | updated_at
timestamp without time zone | user_id
bigint | title
character varying (255) | description
text |
|-------------------|---|---|-------------------|----------------------------------|---|
| 1 | 2025-06-22 20:12:06 | 2025-06-22 20:12:06 | 2 | Software Engineer | As a Software Engineer at Algorix, you will be responsible for d |
| 2 | 2025-06-22 20:12:06 | 2025-06-22 20:12:06 | 4 | Marketing Specialist | Bitwave is seeking a creative and strategic Marketing Specialis |
| 3 | 2025-06-22 20:12:06 | 2025-06-22 20:12:06 | 9 | Web Developer | At NextGen Solutions, our Web Developer will be crucial in buil |
| 4 | 2025-06-22 20:12:06 | 2025-06-22 20:12:06 | 3 | Data Analyst | Quantum Code is in search of a Data Analyst to join our team a |
| 5 | 2025-06-22 20:12:06 | 2025-06-22 20:12:06 | 9 | Graphic Designer | As a Graphic Designer at Shield, you will be at the forefront of |
| 6 | 2025-06-22 20:12:06 | 2025-06-22 20:12:06 | 4 | Data Scientist | Join Sparkle as a Data Scientist and play a pivotal role in analy |
| 7 | 2025-06-22 20:12:06 | 2025-06-22 20:12:06 | 6 | UX Designer | Vencom is seeking a UX Designer to enhance our user experien |
| 8 | 2025-06-22 20:12:06 | 2025-06-22 20:12:06 | 10 | Digital Media Specialist | Digital Media is seeking a Digital Media Specialist to manage a |
| 9 | 2025-06-22 20:12:06 | 2025-06-22 20:12:06 | 3 | Product Manager | Pink Pig is searching for a Product Manager to lead the develo |
| 10 | 2025-06-22 20:12:06 | 2025-06-22 20:12:06 | 6 | IT Support Specialist | Tec Solutions is hiring an IT Support Specialist to provide tech |

| id
[PK] bigint | name
character varying (255) | email
character varying (255) | email_verified_at
timestamp without time zone | password
character varying (255) | remember_token
character varying (255) | created_at
timestamp | updated_at
timestamp without time zone |
|-------------------|---------------------------------|----------------------------------|--|-------------------------------------|---|-------------------------|---|
| 1 | Lonnie Wuckert | layla.fritsch@example.com | 2025-06-22 20:12:05 | \$2y\$12\$Ljg5paVNUfZkv3hZ... | CaE2rC2ffP | 2025-0... | 2025-06-22 20:12:06 |
| 2 | Dr. Angelica Wehner | morar.danielle@example.net | 2025-06-22 20:12:06 | \$2y\$12\$Ljg5paVNUfZkv3hZ... | L1SiCR4aw4 | 2025-0... | 2025-06-22 20:12:06 |
| 3 | Carolyn White | fidel.smitham@example.org | 2025-06-22 20:12:06 | \$2y\$12\$Ljg5paVNUfZkv3hZ... | NGZiPJXjx7 | 2025-0... | 2025-06-22 20:12:06 |
| 4 | Hilda Beer | shanelle.kemmer@example.org | 2025-06-22 20:12:06 | \$2y\$12\$Ljg5paVNUfZkv3hZ... | yJZVU9Fygh | 2025-0... | 2025-06-22 20:12:06 |
| 5 | Bernadine Schultz DDS | jaron.baumbach@example.co... | 2025-06-22 20:12:06 | \$2y\$12\$Ljg5paVNUfZkv3hZ... | rBm7qfmZwR | 2025-0... | 2025-06-22 20:12:06 |
| 6 | Mrs. Alivia Fahey MD | vroberts@example.net | 2025-06-22 20:12:06 | \$2y\$12\$Ljg5paVNUfZkv3hZ... | S9mM9kxDse | 2025-0... | 2025-06-22 20:12:06 |
| 7 | Maximus Schmidt | metz.wilhelm@example.net | 2025-06-22 20:12:06 | \$2y\$12\$Ljg5paVNUfZkv3hZ... | 5wjlgGZanj | 2025-0... | 2025-06-22 20:12:06 |
| 8 | Silas Abshire | wsauer@example.com | 2025-06-22 20:12:06 | \$2y\$12\$Ljg5paVNUfZkv3hZ... | PMNpiCByW5 | 2025-0... | 2025-06-22 20:12:06 |
| 9 | Prof. Domenica Lynch | littel.jovan@example.net | 2025-06-22 20:12:06 | \$2y\$12\$Ljg5paVNUfZkv3hZ... | 7ABgMoCD20 | 2025-0... | 2025-06-22 20:12:06 |
| 10 | Leland Johns | uschmeler@example.org | 2025-06-22 20:12:06 | \$2y\$12\$Ljg5paVNUfZkv3hZ... | FS6IE0GSKU | 2025-0... | 2025-06-22 20:12:06 |

Available Jobs

Software Engineer - As a Software Engineer at Algorix, you will be responsible for designing, developing, and maintaining high-quality software applications. You will work closely with cross-functional teams to deliver scalable and efficient solutions that meet business needs. The role involves writing clean, maintainable code, participating in code reviews, and staying current with industry trends to ensure our technology stack remains cutting-edge.

Marketing Specialist - Bitwave is seeking a creative and strategic Marketing Specialist to develop and execute comprehensive marketing campaigns. In this role, you will be responsible for market research, identifying target audiences, and crafting compelling messages to drive brand awareness and engagement. You'll work closely with the sales and product teams to align marketing strategies with business goals and analyze campaign performance to optimize results.

Web Developer - At NextGen Solutions, our Web Developer will be crucial in building and maintaining exceptional web applications that delight users and meet business objectives. You will be involved in the full development lifecycle, including design, coding, testing, and deployment. The role requires expertise in front-end technologies such as HTML, CSS, and JavaScript, as well as experience with back-end systems to create dynamic and responsive web solutions.

Data Analyst - Quantum Code is in search of a Data Analyst to join our team and transform complex data into actionable insights. You will utilize various analytical tools and techniques to interpret data, identify trends, and provide strategic recommendations. Your role will involve working closely with stakeholders to understand their data needs, delivering detailed reports, and contributing to data-driven decision-making processes to support the company's objectives.

Graphic Designer - As a Graphic Designer at Shield, you will be at the forefront of our creative projects, working on diverse design tasks that range from branding and advertising to digital and print media. You will be responsible for creating visually compelling designs that effectively communicate our brand message and captivate our audience. Collaboration with other

6.13 Note for MySQL/MariaDb users

```
<?php
namespace Database\Seeders;

use App\Models\User;
// use Illuminate\Database\Console\Seeds\WithoutModelEvents;
use Illuminate\Database\Seeder;
use Illuminate\Support\Facades\DB;

class DatabaseSeeder extends Seeder {
    public function run(): void {

        // Disable foreign key checks
        DB::statement('SET FOREIGN_KEY_CHECKS=0;');

        // Truncate tables
        DB::table('job_listings')->truncate();
        DB::table('users')->truncate();

        // Re-enable foreign key checks
        DB::statement('SET FOREIGN_KEY_CHECKS=1;');

        $this->call(RandomUserSeeder::class);
        $this->call(JobSeeder::class);
    }
}
```

If you are using MySQL or MariaDB, you may get an error on the "truncate()" method saying that the foreign key constraint needs to be removed first. If you get this, just remove it, truncate and then re-add it like this:

7. Pages, Presentation & CRUD

[7.1 Intro](#)

[7.2 Job Page & Job Card Component](#)

[7.3 Homepage Jobs](#)

[7.4 Job Details Page](#)

[7.5 Create Job Page](#)

[7.6 Text Input Component](#)

[7.7 Other Input Components](#)

[7.8 Finish Input Validation](#)

[7.9 Flash Messages & Alert Component](#)

[7.10 Alpine.js & Alert dismiss](#)

[7.11 Optional Job Fields](#)

[7.12 File Uploading](#)

[7.13 Update Jobs Listings](#)

[7.14 Delete Job Listing](#)

7.1 Intro

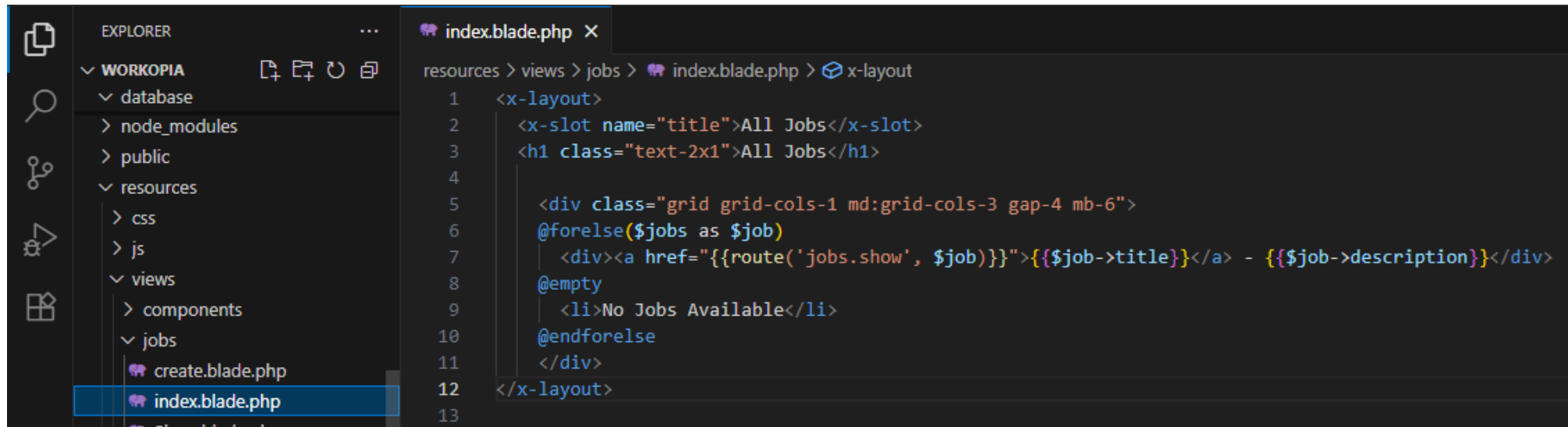


Laravel From Scratch

Page Presentation & CRUD - Section Intro

- ✓ **Job Card Component**
- ✓ **Homepage Jobs**
- ✓ **Details Page**
- ✓ **Create Form**
- ✓ **Update & Delete**
- ✓ **Flash Messages**
- ✓ **Alert Component**
- ✓ **Alpine.js For Interactivity**

7.2 Job Page & Job Card Component



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure for 'WORKOPIA'. The 'resources' folder is expanded, showing 'css', 'js', and 'views'. The 'views' folder is further expanded, showing 'components' and 'jobs'. The 'jobs' folder is selected, and the file 'index.blade.php' is highlighted. The main editor area shows the content of 'index.blade.php' with the following code:

```
resources > views > jobs > index.blade.php > x-layout
1  <x-layout>
2      <x-slot name="title">All Jobs</x-slot>
3      <h1 class="text-2x1">All Jobs</h1>
4
5      <div class="grid grid-cols-1 md:grid-cols-3 gap-4 mb-6">
6          @forelse($jobs as $job)
7              <div><a href="{{route('jobs.show', $job)}}">{{$job->title}}</a> - {{$job->description}}</div>
8          @empty
9              <li>No Jobs Available</li>
10         @endforelse
11     </div>
12 </x-layout>
13
```

The job cards are going to be a separate component blade withing a CSS grid div

```
PS C:\Users\ellio\Herd\workopia> php artisan make:component JobCard

INFO  Component [C:\Users\ellio\Herd\workopia\app\View\Components\JobCard.php] created successfully.

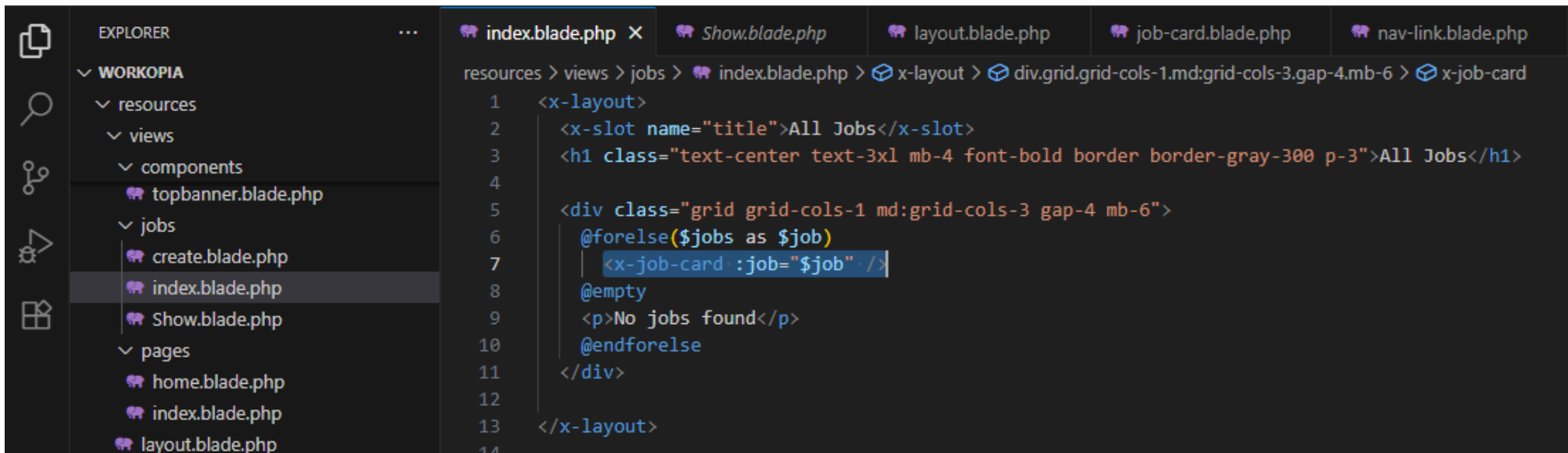
INFO  View [C:\Users\ellio\Herd\workopia\resources\views\components/job-card.blade.php] created successfully.

PS C:\Users\ellio\Herd\workopia>
```

@props(['job'])

@props --> The job card takes in an array of Job

```
<article class="rounded-lg shadow-md bg-white p-4">
  <div class="flex items-center space-between gap-4">
    company_name }}" class="w-14"/>
    <div>
      <h2 class="text-xl font-semibold">{{ $job->title }}</h2>
      <p class="text-sm text-gray-500">{{ $job->job_type }}</p>
    </div>
  </div>
  <p class="text-gray-700 text-lg mt-2">{{ Str::limit($job->description, 100) }}</p>
  <ul class="my-4 bg-gray-100 p-4 rounded">
    <li class="mb-2"><strong>Salary:</strong> {{ number_format($job->salary) }} </li>
    <li class="mb-2"><strong>Location:</strong> {{ $job->city }}, {{ $job->state }}
    @if ($job->remote)
      <span class="text-xs bg-green-500 text-white rounded-full px-2 py-1 ml-2">Remote</span>
    @else
      <span class="text-xs bg-red-500 text-white rounded-full px-2 py-1 ml-2">On-site</span>
    @endif
  </li>
  <li class="mb-2"><strong>Tags:</strong> {{ ucwords(str_replace(',', ' ', $job->tags)) }}</li>
  </ul>
  <a href="{{ route('jobs.show', $job->id) }}" class="block w-full text-center px-5 py-2.5 shadow-sm rounded border text-base font-medium text-indigo-700 bg-indigo-100 hover:bg-indigo-200">Details</a>
</article>
```

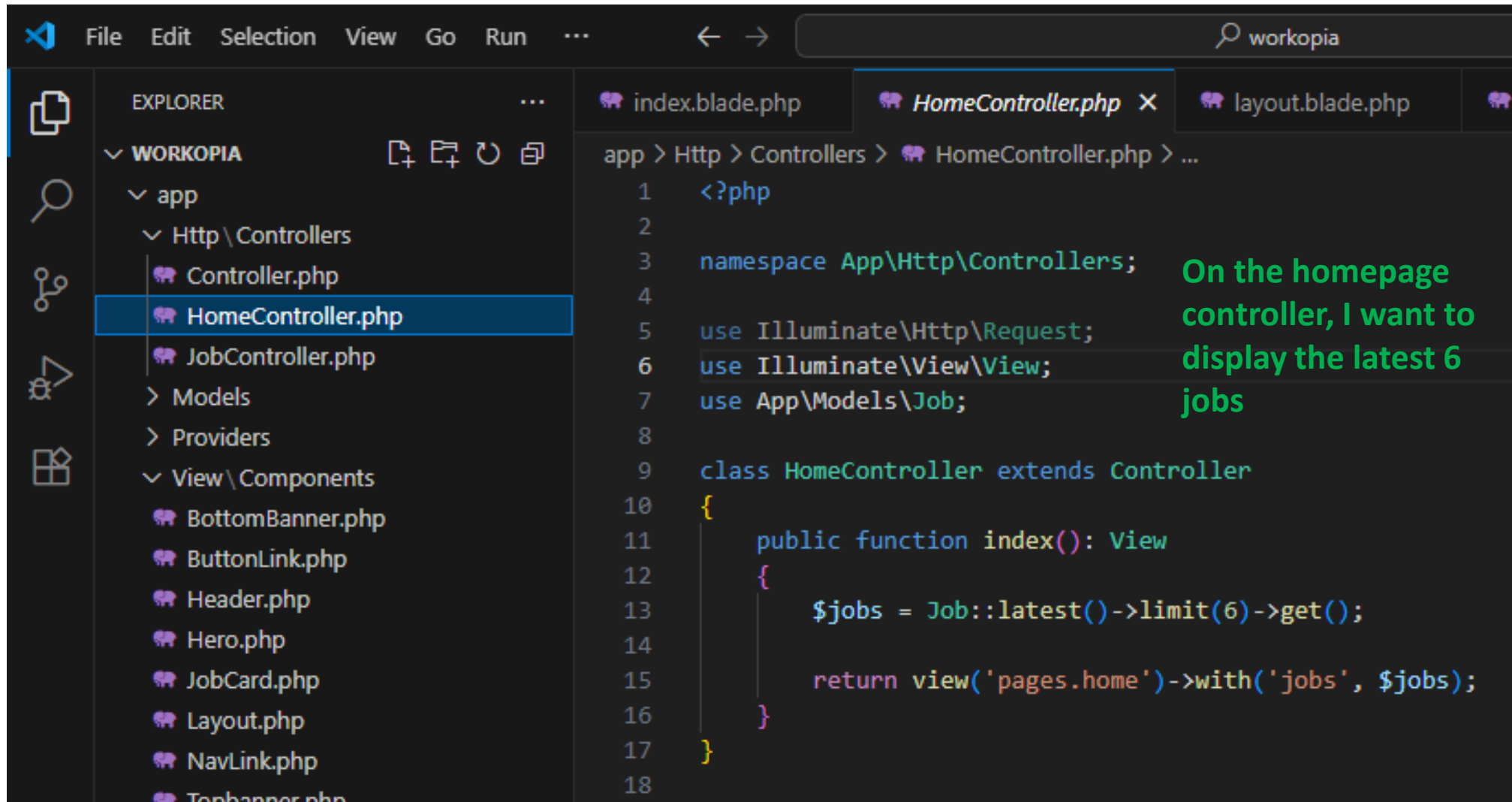
```
<x-layout>
  <x-slot name="title">All Jobs</x-slot>
  <h1 class="text-center text-3xl mb-4 font-bold border border-gray-300 p-3">All
Jobs</h1>
```

```
<div class="grid grid-cols-1 md:grid-cols-3 gap-4 mb-6">
  @forelse($jobs as $job)
    <x-job-card :job="$job" />
  @empty
    <p>No jobs found</p>
  @endforelse
</div>
```

```
</x-layout>
```

Now in jobs/index.blade.php, within the forelse loop, I can use the <x-job-card> component passing in \$job

7.2 Homepage Jobs



The screenshot displays an IDE interface with the following components:

- EXPLORER (Left Panel):** Shows the project structure. The 'app' directory is expanded, showing 'Http' > 'Controllers'. 'HomeController.php' is selected and highlighted in blue. Other files visible include 'Controller.php', 'JobController.php', 'Models', 'Providers', 'View' > 'Components' (containing 'BottomBanner.php', 'ButtonLink.php', 'Header.php', 'Hero.php', 'JobCard.php', 'Layout.php', 'NavLink.php', and 'Topbanner.php').
- File Explorer (Top):** Shows open files: 'index.blade.php', 'HomeController.php' (active), and 'layout.blade.php'.
- Code Editor (Main Area):** Displays the code for 'HomeController.php'. The code is as follows:

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use Illuminate\View\View;
7 use App\Models\Job;
8
9 class HomeController extends Controller
10 {
11     public function index(): View
12     {
13         $jobs = Job::latest()->limit(6)->get();
14
15         return view('pages.home')->with('jobs', $jobs);
16     }
17 }
18
```
- Annotations (Right Side):** Green text annotations are present next to the code:
 - Next to line 3: 'On the homepage controller, I want to display the latest 6 jobs'
 - Next to line 6: 'jobs'

EXPLORER

WORKOPIA

resources

views

componentsheader.blade.phphero.blade.phpjob-card.blade.phpnav-link.blade.phptopbanner.blade.php

jobs

create.blade.phpindex.blade.phpShow.blade.php

pages

home.blade.php

index.blade.php

layout.blade.php

index.blade.php

home.blade.php

layout.blade.php

job-card.blade.php

resources > views > pages > home.blade.php > x-layout

1<x-layout>

2<h2 class="text-center text-3xl mb-4 font-bold border border-gray-300 p-3">

3Recent Jobs

4</h2>

5<div class="grid grid-cols-1 md:grid-cols-3 gap-4 mb-6">

6@forelse(\$jobs as \$job)

7<x-job-card :job="\$job" />

8@empty

9<p>No jobs found</p>

10@endforelse

11</div>

12

13<i class="fa fa-arrow-alt-circle-right"></i> Show All Jobs

14

15<x-bottom-banner />

16</x-layout>

On the homepage view, I can reuse the <-job-card component

And also add a link to the all-jobs page

7.3 Job Details Page

```
<x-layout>
  <div class="grid grid-cols-1 md:grid-cols-4 gap-6">
    <section class="md:col-span-3">
      <div class="rounded-lg shadow-md bg-white p-3">

        <div class="flex justify-between items-center">
          <a class="block p-4 text-blue-700" href="{{ route('jobs.index') }}"><i class="fa fa-arrow-alt-circle-left"></i> Back To
Listings</a>
          <div class="flex space-x-3 ml-4">
            <a href="/edit" class="px-4 py-2 bg-blue-500 hover:bg-blue-600 text-white rounded">Edit</a>
            <!-- Delete Form -->
            <form method="POST">
              <button type="submit" class="px-4 py-2 bg-red-500 hover:bg-red-600 text-white rounded">Delete</button>
            </form>
            <!-- End Delete Form -->
          </div>
        </div>

      </div>

      <div class="p-4">
        <h2 class="text-xl font-semibold">{{$job->title}}</h2>
        <p class="text-gray-700 text-lg mt-2">{{$job->description}}</p>
        <ul class="my-4 bg-gray-100 p-4">
          <li class="mb-2"><strong>Job Type:</strong> {{$job->job_type}}</li>
          <li class="mb-2"><strong>Remote:</strong> {{$job->remote ? 'Yes' : 'No'}}</li>
          <li class="mb-2"><strong>Salary:</strong> {{number_format($job->salary)}}</li>
          <li class="mb-2"><strong>Site Location:</strong> {{$job->city}}, {{$job->state}}</li>
          <li class="mb-2"><strong>Tags:</strong> {{ucwords(str_replace(', ', ', ', $job->tags))}}</li>
        </ul>
      </div>
    </div>
  </div>
</div>
```

```

<div class="container mx-auto p-4">
  <h2 class="text-xl font-semibold mb-4">Job Details</h2>
  <div class="rounded-lg shadow-md bg-white p-4">
    <h3 class="text-lg font-semibold mb-2 text-blue-500">Job Requirements</h3>
    <p>{{ $job->requirements }}</p>
    <h3 class="text-lg font-semibold mt-4 mb-2 text-blue-500">Benefits</h3>
    <p>{{ $job->benefits }}</p>
  </div>
  <p class="my-5">
    Put "Job Application" as the subject of your email and attach your resume.</p>
  <a href="mailto:{{ $job->contact_email }}" class="block w-full text-center px-5 py-2.5 shadow-sm rounded border text-base font-medium cursor-pointer text-indigo-700 bg-indigo-100 hover:bg-indigo-200">Apply Now</a>
</div>

<div class="bg-white p-6 rounded-lg shadow-md mt-6">
  <div id="map"></div>
</div>
</section>

<aside class="bg-white rounded-lg shadow-md p-3">
  <h3 class="text-xl text-center mb-4 font-bold">Company Info</h3>
  company_name }}" class="w-full rounded-lg mb-4 m-auto"/>
  <h4 class="text-lg font-bold">{{ $job->company_name }}</h4>
  <p class="text-gray-700 text-lg my-3">{{ $job->company_description }}</p>
  <a href="{{ $job->company_website }}" target="_blank" class="text-blue-500">Visit Website</a>
  <a href="" class="mt-10 bg-blue-500 hover:bg-blue-600 text-white font-bold w-full py-2 px-4 rounded-full flex items-center justify-center"><i class="fas fa-bookmark mr-3"></i> Bookmark Listing</a>
</aside>

</div>
</x-layout>

```



```

<!-- Job Description -->
<div class="mb-4">
  <label class="block text-gray-700" for="description">Job Description</label>
  <textarea cols="30" rows="7" id="description" name="description"
    class="w-full px-4 py-2 border rounded focus:outline-none
      @error('description')
        border-red-500
      @enderror
    " placeholder="We are seeking a skilled and motivated Software Developer...">{{ old('description')
  }}</textarea>
  @error('description')
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
  @enderror
</div>

<!-- Annual Salary -->
<div class="mb-4">
  <label class="block text-gray-700" for="salary">Annual Salary</label>
  <input id="salary" type="number" name="salary" value="{{ old('salary') }}"
    class="w-full px-4 py-2 border rounded focus:outline-none
      @error('salary')
        border-red-500
      @enderror
    " placeholder="90000" />
  @error('salary')
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
  @enderror
</div>

<!-- Requirements -->
<div class="mb-4">
  <label class="block text-gray-700" for="requirements">Requirements</label>
  <textarea id="requirements" name="requirements"
    class="w-full px-4 py-2 border rounded focus:outline-none @error('requirements')

```



```
<!-- Requirements -->
<div class="mb-4">
  <label class="block text-gray-700" for="requirements">Requirements</label>
  <textarea id="requirements" name="requirements"
    class="w-full px-4 py-2 border rounded focus:outline-none
    @error('requirements')
      border-red-500
    @enderror
    " placeholder="Bachelor's degree in Computer Science">{{ old('requirements') }}</textarea>
  @error('requirements')
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
  @enderror
</div>

<!-- Benefits -->
<div class="mb-4">
  <label class="block text-gray-700" for="benefits">Benefits</label>
  <textarea id="benefits" name="benefits"
    class="w-full px-4 py-2 border rounded focus:outline-none
    @error('benefits')
      border-red-500
    @enderror
    " placeholder="Health insurance, 401k, paid time off">{{ old('benefits') }}</textarea>
  @error('benefits')
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
  @enderror
</div>
```

```

<!-- Tags -->
<div class="mb-4">
  <label class="block text-gray-700" for="tags">Tags (comma-separated)</label>
  <input id="tags" type="text" name="tags" value="{{ old('tags') }}"
    class="w-full px-4 py-2 border rounded focus:outline-none
    @error('tags')
      border-red-500
    @enderror
    " placeholder="development,coding,java,python" />
  @error('tags')
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
  @enderror
</div>

<!-- Job Type -->
<div class="mb-4">
  <label class="block text-gray-700" for="job_type">Job Type</label>
  <select id="job_type" name="job_type"
    class="w-full px-4 py-2 border rounded focus:outline-none
    @error('job_type')
      border-red-500 @enderror">
    <option value="Full-Time" {{ old('job_type') == 'Full-Time' ? 'selected' : '' }}>Full-Time</option>
    <option value="Part-Time" {{ old('job_type') == 'Part-Time' ? 'selected' : '' }}>Part-Time</option>
    <option value="Contract" {{ old('job_type') == 'Contract' ? 'selected' : '' }}>Contract</option>
    <option value="Temporary" {{ old('job_type') == 'Temporary' ? 'selected' : '' }}>Temporary</option>
    <option value="Internship" {{ old('job_type') == 'Internship' ? 'selected' : '' }}>Internship</option>
    <option value="Volunteer" {{ old('job_type') == 'Volunteer' ? 'selected' : '' }}>Volunteer</option>
    <option value="On-Call" {{ old('job_type') == 'On-Call' ? 'selected' : '' }}>On-Call</option>
  </select>
  @error('job_type')
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
  @enderror
</div>

```

```

<!-- Remote -->
<div class="mb-4">
  <label class="block text-gray-700" for="remote">Remote</label>
  <select id="remote" name="remote"
    class="w-full px-4 py-2 border rounded focus:outline-none
    @error('remote')
      border-red-500
    @enderror
    ">
    <option value="0" {{ old('remote')==false ? 'selected' : '' }}>No</option>
    <option value="1" {{ old('remote')==true ? 'selected' : '' }}>Yes</option>
  </select>
  @error('remote')
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
  @enderror
</div>

<h2 class="text-2xl font-bold mb-6 text-center text-gray-500">Company Info</h2>

<!-- Address -->
<div class="mb-4">
  <label class="block text-gray-700" for="address">Address</label>
  <input id="address" type="text" name="address" value="{{ old('address') }}"
    class="w-full px-4 py-2 border rounded focus:outline-none
    @error('address')
      border-red-500
    @enderror
    " placeholder="123 Main St" />
  @error('address')
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
  @enderror
</div>

```



```
<!-- ZIP Code -->
<div class="mb-4">
  <label class="block text-gray-700" for="zipcode">ZIP Code</label>
  <input id="zipcode" type="text" name="zipcode" value="{{ old('zipcode') }}"
    class="w-full px-4 py-2 border rounded focus:outline-none
    @error('zipcode')
      border-red-500
    @enderror
    " placeholder="12201" />
  @error('zipcode')
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
  @enderror
</div>

<!-- Company Name -->
<div class="mb-4">
  <label class="block text-gray-700" for="company_name">Company Name</label>
  <input id="company_name" type="text" name="company_name" value="{{ old('company_name') }}"
    class="w-full px-4 py-2 border rounded focus:outline-none
    @error('company_name')
      border-red-500
    @enderror
    " placeholder="Company name" />
  @error('company_name')
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
  @enderror
</div>
```

```
<!-- Company Description -->
<div class="mb-4">
  <label class="block text-gray-700" for="company_description">Company Description</label>
  <textarea id="company_description" name="company_description"
    class="w-full px-4 py-2 border rounded focus:outline-none
    @error('company_description')
      border-red-500
    @enderror
    " placeholder="Company Description">{{ old('company_description') }}</textarea>
  @error('company_description')
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
  @enderror
</div>

<!-- Company Website -->
<div class="mb-4">
  <label class="block text-gray-700" for="company_website">Company Website</label>
  <input id="company_website" type="url" name="company_website" value="{{ old('company_website') }}"
    class="w-full px-4 py-2 border rounded focus:outline-none
    @error('company_website')
      border-red-500
    @enderror
    " placeholder="Enter website" />
  @error('company_website')
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
  @enderror
</div>
```

```
<!-- Contact Phone -->
<div class="mb-4">
  <label class="block text-gray-700" for="contact_phone">Contact Phone</label>
  <input id="contact_phone" type="text" name="contact_phone" value="{{ old('contact_phone') }}"
    class="w-full px-4 py-2 border rounded focus:outline-none
    @error('contact_phone')
      border-red-500
    @enderror
    " placeholder="Enter phone" />
  @error('contact_phone')
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
  @enderror
</div>

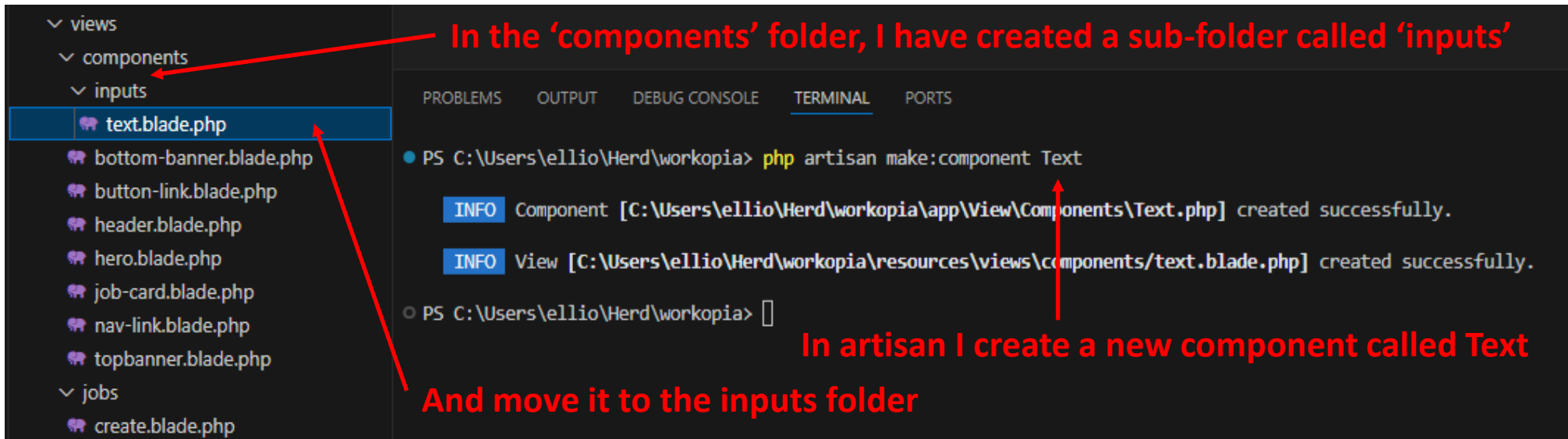
<!-- Contact Email -->
<div class="mb-4">
  <label class="block text-gray-700" for="contact_email">Contact Email</label>
  <input id="contact_email" type="email" name="contact_email" value="{{ old('contact_email') }}"
    class="w-full px-4 py-2 border rounded focus:outline-none
    @error('contact_email')
      border-red-500
    @enderror
    " placeholder="Email where you want to receive applications" />
  @error('contact_email')
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
  @enderror
</div>
```

```
<!-- Company Logo -->
<div class="mb-4">
  <label class="block text-gray-700" for="company_logo">Company Logo</label>
  <input id="company_logo" type="file" name="company_logo"
    class="w-full px-4 py-2 border rounded focus:outline-none
    @error('company_logo')
      border-red-500
    @enderror
    " />
    @error('company_logo')
      <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
    @enderror
  </div>

  <!-- Submit Button -->
  <button type="submit"
    class="w-full bg-green-500 hover:bg-green-600 text-white px-4 py-2 my-3 rounded focus:outline-none">
    Save
  </button>
</form>
<!-- Form End -->

</div>
</x-layout>
```


7.5 Text Input Component



In the 'components' folder, I have created a sub-folder called 'inputs'

And move it to the inputs folder

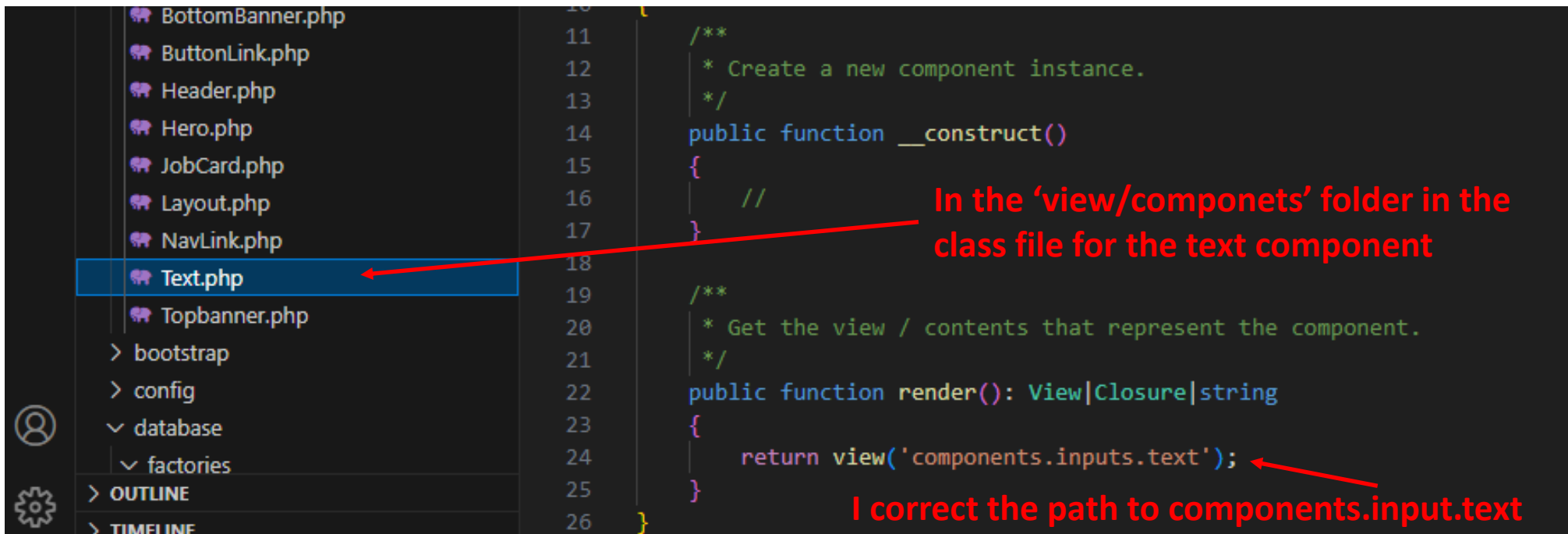
In artisan I create a new component called Text

```
PS C:\Users\ellio\Herd\workopia> php artisan make:component Text
```

INFO Component [C:\Users\ellio\Herd\workopia\app\View\Components\Text.php] created successfully.

INFO View [C:\Users\ellio\Herd\workopia\resources\views\components/text.blade.php] created successfully.

PS C:\Users\ellio\Herd\workopia>



In the 'view/componets' folder in the class file for the text component

I correct the path to components.input.text

```
10
11
12 /**
13  * Create a new component instance.
14  */
15 public function __construct()
16 {
17     //
18 }
19
20 /**
21  * Get the view / contents that represent the component.
22  */
23 public function render(): View|Closure|string
24 {
25     return view('components.inputs.text');
26 }
```

```
@props(['id', 'name', 'label' => null, 'type' => 'text', 'value' => '', 'placeholder' => ''])

<div class="mb-4">
  @if($label)
    <label class="block text-gray-700" for="{{ $id }}">{{ $label }}</label>
  @endif
  <input
    id="{{ $id }}"
    type="{{ $type }}"
    name="{{ $name }}"
    value="{{ old($name, $value) }}"
    class="w-full px-4 py-2 border rounded focus:outline-none @error($name) border-red-500
  @enderror"
    placeholder="{{ $placeholder }}"
  />
  @error($name)
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
  @enderror
</div>
```

The text input blade take in parameters in the
@props that are used to build the html markup

```
bottom-banner.blade.php    5      <!-- Form Start -->
button-link.blade.php      6      <form method="POST" action="{{ route('jobs.store') }}" enctype="multipart/form-data">
header.blade.php           7      @csrf
hero.blade.php             8
job-card.blade.php         9      <h2 class="text-2xl font-bold mb-6 text-center text-gray-500">Job Info</h2>
nav-link.blade.php        10
topbanner.blade.php       11      <!-- Job Title -->
jobs                       12      <x-imput-text id="title" name="title" label="text" placeholder="Software Engineer" />
create.blade.php           13
index.blade.php            14      <div class="mb-4">
Show.blade.php             15      <label class="block text-gray-700" for="title">Job Title</label>
pages                      16      <input id="title" type="text" name="title" value="{{ old('title') }}"
home.blade.php             17      class="w-full px-4 py-2 border rounded focus:outline-none @error('title') border-red-500 @enderror"
index.blade.php            18      placeholder="Software Engineer" />
layout.blade.php           19      @error('title')
welcome.blade.php          20      <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
                           21      @enderror
                           22      </div>
                           23
```

Now the highlighted code can be replaced with

`<x-inputs-text id="title" name="title" label="text" placeholder="Software Engineer"/>`

```
<!-- Annual Salary -->
<x-inputs-text id="salary" name="salary" type="number" label="salary" placeholder="90000" />

<div class="mb-4">
    <label class="block text-gray-700" for="salary">Annual Salary</label>
    <input id="salary" type="number" name="salary" value="{{ old('salary') }}"
        class="w-full px-4 py-2 border rounded focus:outline-none @error('salary') border-red-500 @enderror"
        placeholder="90000" />
    @error('salary')
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
    @enderror
</div>
```

The text input blade can be reused for salary input html passing the type as number

```

50      <!-- Tags -->
51      <x-inputs.text id="tags" name="tags" label="tags" placeholder="development,coding,java,python" />
52
53      <div class="mb-4">
54          <label class="block text-gray-700" for="tags">Tags (comma-separated)</label>
55          <input id="tags" type="text" name="tags" value="{{ old('tags') }}"
56              class="w-full px-4 py-2 border rounded focus:outline-none @error('tags') border-red-500 @enderror"
57              placeholder="development,coding,java,python" />
58          @error('tags')
59              <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
60          @enderror
61      </div>

```

And for the tags input the component can be reused.

```

86      <!-- Address -->
87      <x-inputs.text id="address" name="address" label="address" placeholder="123 Main St" />
88
89      <div class="mb-4">
90          <label class="block text-gray-700" for="address">Address</label>
91          <input id="address" type="text" name="address" value="{{ old('address') }}"
92              class="w-full px-4 py-2 border rounded focus:outline-none @error('address') border-red-500 @enderror"
93              placeholder="123 Main St" />
94          @error('address')
95              <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
96          @enderror
97      </div>

```

And for the address input the component can be reused.

```

89     <!-- City -->
90     <x-inputs.text id="city" name="city" label="city" placeholder="Albany" />
91
92     <div class="mb-4">
93         <label class="block text-gray-700" for="city">City</label>
94         <input id="city" type="text" name="city" value="{{ old('city') }}"
95             class="w-full px-4 py-2 border rounded focus:outline-none @error('city') border-red-500 @enderror"
96             placeholder="Albany" />
97         @error('city')
98             <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
99         @enderror
100     </div>

```

And for the city input the component can be reused.

```

<!-- State -->
<x-inputs.text id="state" name="state" label="state" placeholder="NY" />

<div class="mb-4">
    <label class="block text-gray-700" for="state">State</label>
    <input id="state" type="text" name="state" value="{{ old('state') }}"
        class="w-full px-4 py-2 border rounded focus:outline-none @error('state') border-red-500 @enderror"
        placeholder="NY" />
    @error('state')
        <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
    @enderror
</div>

```

And for the state input the component can be reused.

```

95     <!-- ZIP Code -->
96     <x-inputs.text id="zipcode" name="zipcode" label="zipcode" placeholder="12201" />
97
98     <div class="mb-4">
99         <label class="block text-gray-700" for="zipcode">ZIP Code</label>
100         <input id="zipcode" type="text" name="zipcode" value="{{ old('zipcode') }}"
101             class="w-full px-4 py-2 border rounded focus:outline-none @error('zipcode') border-red-500 @enderror"
102             placeholder="12201" />
103         @error('zipcode')
104             <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
105         @enderror
106     </div>

```

And for the zipcode input the component can be reused.

```

98     <!-- Company Name -->
99     <x-inputs.text id="company_name" name="company_name" label="company_name" placeholder="Company name" />
100
101     <div class="mb-4">
102         <label class="block text-gray-700" for="company_name">Company Name</label>
103         <input id="company_name" type="text" name="company_name" value="{{ old('company_name') }}"
104             class="w-full px-4 py-2 border rounded focus:outline-none @error('company_name') border-red-500
105             @enderror"
106             placeholder="Company name" />
107         @error('company_name')
108             <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
109         @enderror
110     </div>

```

And for the company name input the component can be reused.

```
<!-- Company Website -->
<x-inputs.text id="company_website" name="company_website" type="url" label="company_website"
placeholder="Enter website" />
```

```
.....<div class="mb-4">
.....<label class="block text-gray-700" for="company_website">Company Website</label>
.....<input id="company_website" type="url" name="company_website" value="{{ old('company_website') }}"
.....class="w-full px-4 py-2 border rounded focus:outline-none @error('company_website') border-red-500
.....@enderror"
.....placeholder="Enter website" />
.....@error('company_website')
.....<p class="text-red-500 text-sm mt-1">{{ $message }}</p>
.....@enderror
.....</div>
```

And for the company website input the component can be reused specifying a type of url.

```
115 <!-- Contact Phone -->
116 <x-inputs.text id="contact_phone" name="contact_phone" label="Contact Phone" placeholder="Enter website" />
117
118 ✓ .....<div class="mb-4">
119 .....<label class="block text-gray-700" for="contact_phone">Contact Phone</label>
120 ✓ .....<input id="contact_phone" type="text" name="contact_phone" value="{{ old('contact_phone') }}"
121 .....class="w-full px-4 py-2 border rounded focus:outline-none @error('contact_phone') border-red-500
122 .....@enderror"
122 .....placeholder="Enter phone" />
123 ✓ .....@error('contact_phone')
124 .....<p class="text-red-500 text-sm mt-1">{{ $message }}</p>
125 .....@enderror
126 .....</div>
127
```

And for the Contact_phone input the component can be replaced

Activate Window

```

118 <!-- Contact Email -->
119 <x-inputs.text id="contact_email" name="contact_email" type="email" label="Contact Email" placeholder="Enter
    website" />
120
121 <div class="mb-4">
122     <label class="block text-gray-700" for="contact_email">Contact Email</label>
123     <input id="contact_email" type="email" name="contact_email" value="{{ old('contact_email') }}"
124         class="w-full px-4 py-2 border rounded focus:outline-none @error('contact_email') border-red-500
            @enderror"
125         placeholder="Email where you want to receive applications" />
126     @error('contact_email')
127         <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
128     @enderror
129 </div>

```

And for the Email input the component can be replaced specifying a type as email

7.6 Other Input Components

```
@props(['id', 'name', 'label' => null, 'value' => '', 'placeholder' => ''])

<div class="mb-4">
  @if($label)
    <label class="block text-gray-700" for="{{ $id }}">{{ $label }}</label>
  @endif
  <textarea
    id="{{ $id }}"
    name="{{ $name }}"
    cols="30"
    rows="7"
    class="w-full px-4 py-2 border rounded focus:outline-none @error($name) border-
red-500 @enderror"
    placeholder="{{ $placeholder }}"
  >
    {{ old($name, $value) }}</textarea>
  >
  @error($name)
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
  @enderror
</div>
```

I create a new blade in compinents.inpusts for a text area input and build the html and “props parameters. I also modify the path in the class file.

```

14 <!-- Job Description -->
15 <x-inputs.textarea id="description" name="description" label="Job Description" placeholder="We are seeking a
    skilled and motivated Software Developer..." />
16
17 <div class="mb-4">
18     <label class="block text-gray-700" for="description">Job Description</label>
19     <textarea cols="30" rows="7" id="description" name="description"
20         class="w-full px-4 py-2 border rounded focus:outline-none @error('description') border-red-500 @enderror"
21         placeholder="We are seeking a skilled and motivated Software Developer...">{{ old('description') }}</
        textarea>
22     @error('description')
23         <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
24     @enderror
25 </div>
26

```

Now I can reuse this text area input blade for the description input

```

19 <!-- Requirements -->
20
21 <x-inputs.textarea id="requirements" name="requirements" label="Requirements" placeholder="Bachelor's degree
    in Computer Science" />
22
Users\ellio\Herd\workopia\resources\views\components\topbanner.blade.php
24 <label class="block text-gray-700" for="requirements">Requirements</label>
25 <textarea id="requirements" name="requirements"
26     class="w-full px-4 py-2 border rounded focus:outline-none @error('requirements') border-red-500
        @enderror"
27     placeholder="Bachelor's degree in Computer Science">{{ old('requirements') }}</textarea>
28 @error('requirements')
29     <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
30 @enderror
31 </div>
32

```

And reuse for Requirements input

```

<!-- Benefits -->
<x-inputs.textarea id="benefits" name="benefits" label="Benefits" placeholder="Health insurance, 401k, paid
time off" />

<div class="mb-4">
  <label class="block text-gray-700" for="benefits">Benefits</label>
  <textarea id="benefits" name="benefits"
    class="w-full px-4 py-2 border rounded focus:outline-none @error('benefits') border-red-500 @enderror"
    placeholder="Health insurance, 401k, paid time off">{{ old('benefits') }}</textarea>
    @error('benefits')
  <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
  @enderror
</div>

```

And reuse for Benefits input

```

77 <!-- Company Description -->
78 <x-inputs.textarea id="company_description" name="company_description" label="Company Description"
placeholder="Company Description" />
79
80 <div class="mb-4">
81   <label class="block text-gray-700" for="company_description">Company Description</label>
82   <textarea id="company_description" name="company_description"
83     class="w-full px-4 py-2 border rounded focus:outline-none @error('company_description') border-red-500
      @enderror"
      placeholder="Company Description">{{ old('company_description') }}</textarea>
84   @error('company_description')
85   <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
86   @enderror
87 </div>
88
89

```

And reuse for Company Description input

```

@props(['id', 'name', 'label' => null, 'options' => [], 'value' => ''])

<div class="mb-4">
  @if($label)
    <label class="block text-gray-700" for="{{ $id }}">{{ $label }}</label>
  @endif
  <select id="{{ $id }}" name="{{ $name }}"
    class="w-full px-4 py-2 border rounded focus:outline-none @error($name) border-red-500
@enderror">
    @foreach($options as $optionValue => $optionLabel)
      <option value="{{ $optionValue }}" {{ old($name, $value)==$optionValue ? 'selected' : ''
    }}>
        {{ $optionLabel }}
      </option>
    @endforeach
  </select>
  @error($name)
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
  @enderror
</div>

```

I create a new blade in compinents.inpusts for a select input and build the html and “props parameters. Note that the options prop takes an array because there may be more than one select option. I also modify the path in the class file.

```

<!-- Job Type -->
<x-inputs.select id="job_type" name="job_type" label="Job Type" value="{{old('job_type')}}" :options="
['Full-Time' => 'Full-Time', 'Part-Time' => 'Part-Time', 'Contract' => 'Contract', 'Temporary' =>
'Temporary', 'Internship' => 'Internship', 'On-Call' => 'On-Call']" />

<div class="mb-4">
    <label class="block text-gray-700" for="job_type">Job Type</label>
    <select id="job_type" name="job_type"
        class="w-full px-4 py-2 border rounded focus:outline-none @error('job_type') border-red-500 @enderror">
        <option value="Full-Time" {{ old('job_type') == 'Full-Time' ? 'selected' : '' }}>Full-Time</option>
        <option value="Part-Time" {{ old('job_type') == 'Part-Time' ? 'selected' : '' }}>Part-Time</option>
        <option value="Contract" {{ old('job_type') == 'Contract' ? 'selected' : '' }}>Contract</option>
        <option value="Temporary" {{ old('job_type') == 'Temporary' ? 'selected' : '' }}>Temporary</option>
        <option value="Internship" {{ old('job_type') == 'Internship' ? 'selected' : '' }}>Internship</option>
        <option value="Volunteer" {{ old('job_type') == 'Volunteer' ? 'selected' : '' }}>Volunteer</option>
        <option value="On-Call" {{ old('job_type') == 'On-Call' ? 'selected' : '' }}>On-Call</option>
    </select>
    @error('job_type')
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
    @enderror
</div>

```

As before, I can replace the html with the reusable x-inputs-select component. Note that I am passing in an array for options and using the old Laravel function to display a default value.

```

<!-- Remote -->
<x-inputs.select id="remote" name="remote" label="Remote" :options="['0' => 'No', '1' => 'Yes']" />

<div class="mb-4">
  <label class="block text-gray-700" for="remote">Remote</label>
  <select id="remote" name="remote"
    class="w-full px-4 py-2 border rounded focus:outline-none @error('remote') border-red-500 @enderror">
    <option value="0" {{ old('remote')==false ? 'selected' : '' }}>No</option>
    <option value="1" {{ old('remote')==true ? 'selected' : '' }}>Yes</option>
  </select>
  @error('remote')
  <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
  @enderror
</div>

```

Reusing for the remote selection. I am using 0 and 1 with No and Yes in the array

```
@props(['id', 'name', 'label' => null])
```

```
<div class="mb-4">
```

```
  @if($label)
```

```
    <label class="block text-gray-700" for="{{ $id }}">{{ $label }}</label>
```

```
  @endif
```

```
  <input
```

```
    id="{{ $id }}"
```

```
    type="file"
```

```
    name="{{ $name }}"
```

```
    class="w-full px-4 py-2 border rounded focus:outline-none @error($name) border-red-500
```

```
@enderror"
```

```
  />
```

```
  @error($name)
```

```
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
```

```
  @enderror
```

```
</div>
```

I create a new blade in components.inputs for a file input and build the html and "props parameters. I also modify the path in the class file.

```
65 <!-- Company Logo -->
```

```
66 <x-inputs.file id="company_logo" name="company_logo" label="Company Logo" />
```

```
67
```

```
68 <div class="mb-4">
```

```
69   <label class="block text-gray-700" for="company_logo">Company Logo</label>
```

```
70   <input id="company_logo" type="file" name="company_logo"
```

```
71     class="w-full px-4 py-2 border rounded focus:outline-none @error('company_logo') border-red-500
```

```
    @enderror" />
```

```
72   @error('company_logo')
```

```
73     <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
```

```
74   @enderror
```

```
75 </div>
```

```
76
```

And use it passing in the parameters.

7.7 Finish input Validation

```
public function store(Request $request): RedirectResponse {
```

```
    $validatedData = $request->validate([
        'title' => 'required|string|max:255',
        'description' => 'required|string',
        'salary' => 'required|integer',
        'tags' => 'nullable|string',
        'job_type' => 'required|string',
        'remote' => 'required|boolean',
        'requirements' => 'nullable|string',
        'benefits' => 'nullable|string',
        'address' => 'nullable|string',
        'city' => 'required|string',
        'state' => 'required|string',
        'zipcode' => 'required|string',
        'contact_email' => 'required|email',
        'contact_phone' => 'nullable|string',
        'company_name' => 'required|string',
        'company_description' => 'nullable|string',
        'company_logo' => 'nullable|image|mimes:jpeg,png,jpg,gif|max:2048',
        'company_website' => 'nullable|url'
    ]);
```

```
// Temporarily Hardcode the user_id for new job inputs
$validatedData['user_id'] = 1;
```

```
Job::create($validatedData);
```

```
return redirect()->route('jobs.index')->with('success', 'Job listing created Sucessfully!');
```

```
}
```

I define the validation criteria for each field of the input form

Note for the company logo I am restricting the input file type to jpg, png, jpeg or gif with a max size of 2048 kb.

For the moment the user that is creating the job is hardcoded. I will fix this later.

I have also added a success message

7.8 Flash Messages & Alert Component

```
return redirect()->route('jobs.index')->with('success', 'Job listing created Sucessfully!');
```

When I use the with method, I am creating a temporary session which will be available to me via a session helper so that I can use it on the next page

The screenshot shows the Visual Studio Code interface with the following components:

- EXPLORER:** The file explorer on the left shows the project structure. The 'components' folder under 'views' is expanded, and 'alert.blade.php' is selected, indicated by a red arrow.
- EDITOR:** The main editor area shows the content of 'alert.blade.php'. It contains a Blade component definition with a single parameter 'message'. The component body uses a conditional statement to display a flash message with a green background for success and a red background for error. The code is as follows:

```
@props(['type', 'message'])

@if(session()->has('success') || session()->has('error'))
<div class="p-4 mb-4 text-sm text-white {{ $type === 'success' ? 'bg-green-500' : 'bg-red-500' }}" rounded">
    {{ $message }}
</div>
@endif
```
- TERMINAL:** The terminal at the bottom shows the command to create the component and the successful output messages.

```
PS C:\Users\ellio\Herd\workopia> php artisan make:component Alert

[INFO] Component [C:\Users\ellio\Herd\workopia\app\View\Components\Alert.php] created successfully.
[INFO] View [C:\Users\ellio\Herd\workopia\resources\views\components\alert.blade.php] created successfully.
```

A red arrow points from the terminal output to the text "Create an Alert component in artisan".

```

<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.7.2/css/all.min.css" integrity="sha512-Evv84Mr4kqVGRNSgIGL/F/aIDqQb7xQ2vcrdIwxfjThSH8CSR7PBEakCr51Ck+w+/U6swU2Im1vVX0SVk9ABhg==" crossorigin="anonymous" referrerpolicy="no-referrer" />
  <script src="https://cdn.jsdelivr.net/npm/@tailwindcss/browser@4"></script>
  <link rel="stylesheet" href="{{ asset('css/style.css') }}" />
  <title>{{$title ?? 'Workopia | find and list jobs'}}</title>
</head>

<body class="bg-gray-100">
  <x-header/>
  @if(request()->is('/'))
  <x-hero/>
  <x-top-banner />
  @endif
  <main class="container-mx-auto p-4 mt-4">
    {{-- Display Alert Message --}}
    @if(session('success'))
      <x-alert type="success" message="{{session('success')}}" />
    @endif
    @if(session('error'))
      <x-alert type="error" message="{{session('error')}}" />
    @endif
    {{$slot}}
  </main>
  <script src="{{ asset('js/script.js') }}"></script>
</body>

</html>

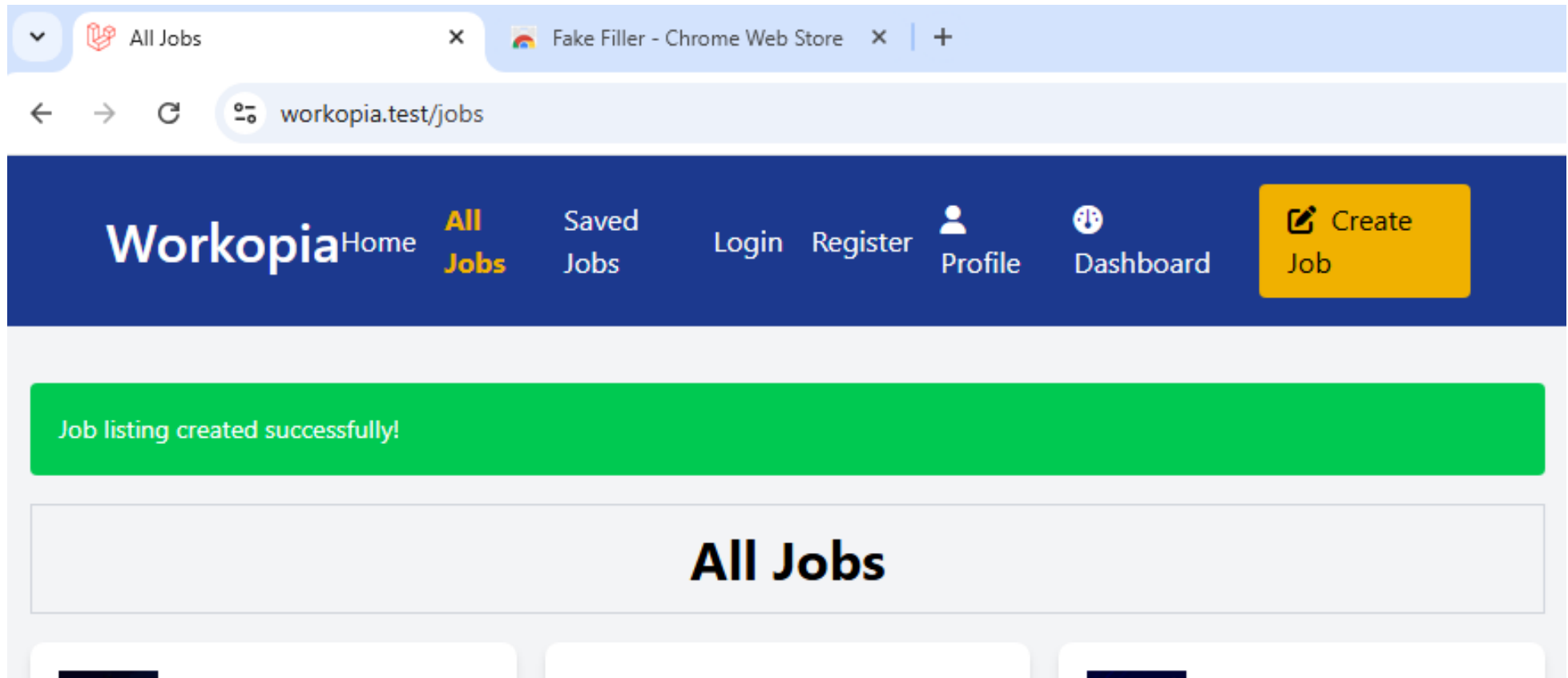
```

Now on the layout page I can access the error message via the session variable and if it is success I can display the success message, and I can also do the same if it is error



This is on the layout page so that it does not have to be repeated in all the other blades.

The message will be at the top of the main container



I can use the chrome extension 'fake filler' to add dummy values to the create job form and when I click success and the 'all jobs' page is loaded, I get the success message.

7.10 Alpine.JS & Alert Dismiss

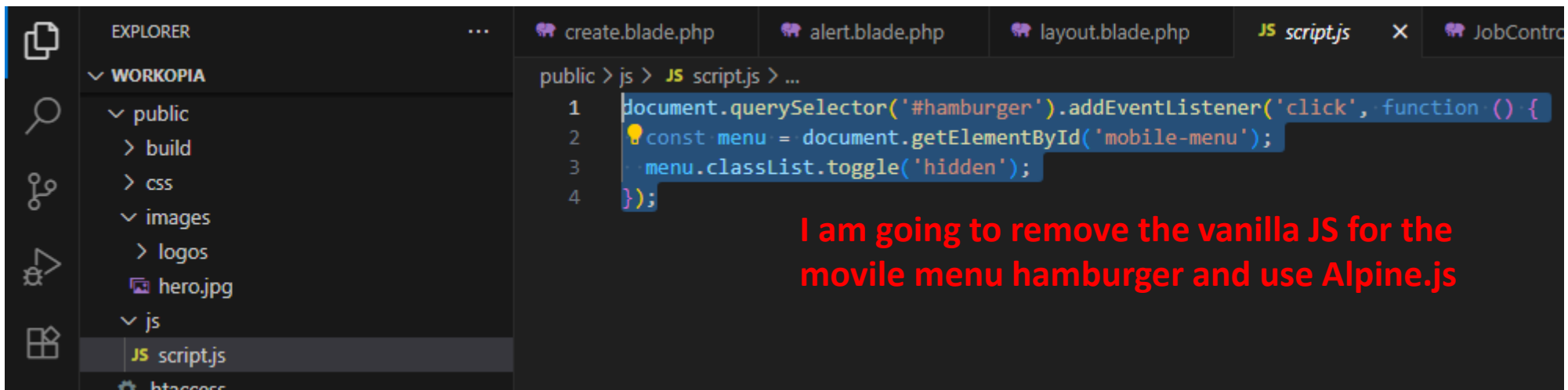
```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/tailwindcss@2.2.19/dist/tailwind.min.css">
  <script src="//unpkg.com/alpinejs" defer></script> ←
  <script src="https://cdn.jsdelivr.net/npm/@tailwindcss/browser@4"></script>
  <link rel="stylesheet" href="{{ asset('css/style.css') }}" />
  <title>{{ $title ?? 'Workopia | find and list jobs' }}</title>
</head>
```

<https://alpinejs.dev>

```
@props(['type', 'message', 'timeout' => '5000'])

<div
  x-data="{ show: true }"
  x-init="setTimeout(() => show = false, {{ timeout }})"
  x-show="show"
  class="p-4 mb-4 text-sm text-white {{ $type === 'success' ? 'bg-green-500' : 'bg-red-500' }} rounded"
  {{ session($type) }}
</div>
```

Alpine JS uses x-data to detect elements so I can modify the alert to be viable at first and set a timeout so that it disappears after a default of 5000ms.



```

<header class="bg-blue-900 text-white p-4" x-data="{ open: false }">
  <div class="container mx-auto flex justify-between items-center">
    <h1 class="text-3xl font-semibold">
      <x-nav-link url="/" :active="request()->is('/')">Workopia</x-nav-link>
    </h1>
    <nav class="hidden md:flex items-center space-x-4">
      <x-nav-link url="/" :active="request()->is('/')">Home</x-nav-link>
      <x-nav-link url="/jobs" :active="request()->is('jobs')">All Jobs</x-nav-link>
      <x-nav-link url="/jobs/saved" :active="request()->is('jobs/saved')">Saved Jobs</x-nav-link>
      <x-nav-link url="/login" :active="request()->is('login')">Login</x-nav-link>
      <x-nav-link url="/register" :active="request()->is('register')">Register</x-nav-link>
      <x-nav-link url="/profile" :active="request()->is('profile')> Profile</x-nav-link>
      <x-nav-link url="/dashboard" :active="request()->is('profile')> Dashboard</x-nav-link>
      <x-button-link url="/jobs/create" icon="edit">Create Job</x-button-link>
    </nav>
    <button @click="open = !open" class="text-white md:hidden flex items-center">
      <i class="fa fa-bars text-2xl"></i>
    </button>
  </div>
  <!-- Mobile Menu -->
  <nav
    x-show="open"
    @click.away="open = false"
    class="md:hidden bg-blue-900 text-white mt-5 pb-4 space-y-2"
  >
    <x-nav-link url="/jobs" :active="request()->is('jobs')> All Jobs</x-nav-link>
    <x-nav-link url="/jobs/saved" :active="request()->is('jobs/saved')> Saved Jobs</x-nav-link>
    <x-nav-link url="/login" :active="request()->is('login')> Login</x-nav-link>
    <x-nav-link url="/register" :active="request()->is('register')> Register</x-nav-link>
    <x-nav-link url="/dashboard" :active="request()->is('dashboard')> Dashbaord</x-nav-link>
    <x-button-link url="/jobs/create" :block="true" icon="edit"> Create Job</x-button-link>
  </nav>
</header>

```

**Set default open
state to false**

**Set hamburger the
button to fire on click**

**Set the mobile nav to
show when state is open**

**Click away will close the menu when the
page is clicked outside of the mobile nav**

```
@props(['job'])
```

```
<article class="rounded-lg shadow-md bg-white p-4">
  <div class="flex items-center space-between gap-4">
    @if($job->company_logo)
      company_name }}"
        class="w-14"
      />
    @endif
    <div>
      <h2 class="text-xl font-semibold">{{ $job->title }}</h2>
      <p class="text-sm text-gray-500">{{ $job->job_type }}</p>
    </div>
  </div>
  <p class="text-gray-700 text-lg mt-2">{{ Str::limit($job->description, 100) }}</p>
  <ul class="my-4 bg-gray-100 p-4 rounded">
    <li class="mb-2"><strong>Salary:</strong> {{ number_format($job->salary) }} </li>
    <li class="mb-2"><strong>Location:</strong> {{ $job->city }}, {{ $job->state }}
    @if ($job->remote)
      <span class="text-xs bg-green-500 text-white rounded-full px-2 py-1 ml-2">Remote</span>
    @else
      <span class="text-xs bg-red-500 text-white rounded-full px-2 py-1 ml-2">On-site</span>
    @endif
  </li>
    @if ($job->tags)
      <li class="mb-2">
        <strong>Tags:</strong>
        {{ ucwords(str_replace(', ', ' ', $job->tags)) }}
      </li>
    @endif
  </ul>
  <a href="{{ route('jobs.show', $job->id) }}" class="block w-full text-center px-5 py-2.5 shadow-sm rounded
border text-base font-medium text-indigo-700 bg-indigo-100 hover:bg-indigo-200">Details</a>
</article>
```

7.11 Optional Job Fields On the cards

The logo is optional when creating a new job so I will wrap it in @if so that it only shows if there is a logo

The tags are optional when creating a new job so I will wrap it in @if so that it only shows if there are tags

7.11 Optional Job Fields On the show.blade

```
<x-layout>
  <div class="grid grid-cols-1 md:grid-cols-4 gap-6">
    <section class="md:col-span-3">
      <div class="rounded-lg shadow-md bg-white p-3">
        <div class="flex justify-between items-center">
          <a class="block p-4 text-blue-700" href="{{ route('jobs.index') }}">
            <i class="fa fa-arrow-alt-circle-left"></i>
            Back To Listings
          </a>
          <div class="flex space-x-3 ml-4">
            <a
              href="/edit"
              class="px-4 py-2 bg-blue-500 hover:bg-blue-600 text-white rounded"
            >Edit</a>
          >
          <!-- Delete Form -->
          <form method="POST">
            <button
              type="submit"
              class="px-4 py-2 bg-red-500 hover:bg-red-600 text-white rounded"
            >
              Delete
            </button>
          </form>
          <!-- End Delete Form -->
        </div>
      </div>
    </div>
    <div class="p-4">
      <h2 class="text-xl font-semibold">{{$job->title}}</h2>
      <p class="text-gray-700 text-lg mt-2">{{$job->description}}</p>
      <ul class="my-4 bg-gray-100 p-4">
        <li class="mb-2"><strong>Job Type:</strong> {{$job->job_type}}</li>
        <li class="mb-2">
          <strong>Remote:</strong> {{$job->remote ? 'Yes' : 'No'}}
        </li>
        <li class="mb-2">
```



```
        <strong>Salary:</strong> {{ number_format($job->salary) }}
    </li>
    <li class="mb-2">
        <strong>Site Location:</strong> {{ $job->city }}, {{ $job->state }}
    </li>
    @if ($job->tags)
    <li class="mb-2">
        <strong>Tags:</strong>
        {{ ucwords(str_replace(', ', ' ', $job->tags)) }}
    </li>
    @endif
</ul>
</div>
</div>
```

```
<div class="container mx-auto p-4">
    @if ($job->requirements || $job->benefits)
    <h2 class="text-xl font-semibold mb-4">Job Details</h2>
    <div class="rounded-lg shadow-md bg-white p-4">
        @if ($job->requirements)
        <h3 class="text-lg font-semibold mb-2 text-blue-500">
            Job Requirements
        </h3>
        <p>{{ $job->requirements }}</p>
        @endif
        @if ($job->benefits)
        <h3 class="text-lg font-semibold mt-4 mb-2 text-blue-500">
            Benefits
        </h3>
        <p>{{ $job->benefits }}</p>
        @endif
    </div>
    @endif
    <p class="my-5">
        Put "Job Application" as the subject of your email and attach your
        resume.
    </p>
```

```
<a
  href="mailto:{{$job->contact_email}}"
  class="block w-full text-center px-5 py-2.5 shadow-sm rounded border text-base font-medium cursor-
pointer text-indigo-700 bg-indigo-100 hover:bg-indigo-200"
>
  Apply Now
</a>
</div>
```

```
<div class="bg-white p-6 rounded-lg shadow-md mt-6">
  <div id="map"></div>
</div>
</section>
```

```
<aside class="bg-white rounded-lg shadow-md p-3">
  <h3 class="text-xl text-center mb-4 font-bold">Company Info</h3>
  @if ($job->company_logo)
    company_name}}"
      class="w-full rounded-lg mb-4 m-auto"
    />
  @endif
  @if ($job->company_name)
    <h4 class="text-lg font-bold">{{$job->company_name}}</h4>
  @endif
  @if ($job->company_description)
    <p class="text-gray-700 text-lg my-3">{{$job->company_description}}</p>
  @endif
  @if ($job->company_website)
    <a href="{{$job->company_website}}" target="_blank" class="text-blue-500"
      >Visit Website</a>
  >
  @endif
  @if ($job->company_phone)
    <p class="text-gray-700 text-lg mt-2">Phone: {{$job->company_phone}}</p>
  @endif
```

```
<a
  href=""
  class="mt-10 bg-blue-500 hover:bg-blue-600 text-white font-bold w-full py-2 px-4 rounded-full flex
items-center justify-center"
  ><i class="fas fa-bookmark mr-3"></i> Bookmark Listing</a
>
</aside>
</div>
</x-layout>
```

7.12 File Uploading

The screenshot shows an IDE with the following structure:

- EXPLORER
 - WORKOPIA
 - app
 - Http \ Controllers
 - Controller.php
 - HomeController.php
 - JobController.php
 - Models
 - Providers
 - View \ Components
 - Alert.php
 - BottomBanner.php
 - ButtonLink.php
 - file.php
 - Header.php
 - Hero.php
 - JobCard.php
 - Layout.php

The main editor shows the `JobController.php` file with the following code:

```
app > Http > Controllers > JobController.php > JobController > store
10  class JobController extends Controller
21  {
22
23      public function store(Request $request): RedirectResponse
24      {
25          dd(request->file('company_logo'));
26          dd($request->file('company_logo'));
27
28          $validatedData = $request->validate([
29              'title' => 'required|string|max:255',
30              'description' => 'required|string',
31              'salary' => 'required|integer',
32              'tags' => 'nullable|string',
33              'job_type' => 'required|string',
34              'remote' => 'required|boolean',
35              'requirements' => 'nullable|string',
36              'benefits' => 'nullable|string',
37              'address' => 'nullable|string',
38              'city' => 'required|string',
```

Dd is a die and dump method in Laravel

```
Illuminate\Http\UploadedFile {#1230 ▼ // app\Http\Controllers\JobController.php:26
```

```
-originalName: "logo-algorix.png"  
-mimeType: "image/png"  
-error: 0  
-originalPath: "logo-algorix.png"  
-test: false  
#hashName: null  
path: "C:\Users\ellio\AppData\Local\Temp"  
filename: "phpE912.tmp"  
basename: "phpE912.tmp"  
pathname: "C:\Users\ellio\AppData\Local\Temp\phpE912.tmp"  
extension: "tmp"  
realPath: "C:\Users\ellio\AppData\Local\Temp\phpE912.tmp"  
aTime: 2025-06-24 11:27:08  
mTime: 2025-06-24 11:27:08  
cTime: 2025-06-24 11:27:08  
inode: 844424930454024  
size: 14791  
perms: 0100666  
owner: 0  
group: 0  
type: "file"  
writable: true  
readable: true  
executable: false  
file: true  
dir: false  
link: false  
linkTarget: "C:\Users\ellio\AppData\Local\Temp\phpE912.tmp" }  
}
```

Now, when I use the create.blade and input test values from fake filler including a dummy file upload, the dd (die and dump) lands on this splash screen with diagnostic info

Dd gives diagnostic data about the file such as original name, mime type, original path etc

I have to add a symlink so that files that are uploaded are available to the public. i.e. from a folder called storage\App\public

The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure under 'WORKOPIA'. The 'storage' folder is expanded, showing 'app' and 'public' subfolders. A red arrow points from the text above to the 'public' folder. The main editor displays the 'JobController.php' file, showing the 'create' and 'store' methods. The 'store' method uses 'dd(\$request->file('company_logo'))'. At the bottom, the Terminal panel shows the command 'php artisan storage:link' being executed, followed by an informational message: 'The [C:\Users\ellio\Herd\workopia\public\storage] link has been connected to [C:\Users\ellio\Herd\workopia\storage\app\public].'

```
app > Http > Controllers > JobController.php > JobController > store
10  class JobController extends Controller
18  public function create(): View
19  {
20      return view('jobs.create');
21  }
22
23  public function store(Request $request): RedirectResponse
24  {
25
26      dd($request->file('company_logo'));
27  }
```

PS C:\Users\ellio\Herd\workopia> php artisan storage:link

INFO The [C:\Users\ellio\Herd\workopia\public\storage] link has been connected to [C:\Users\ellio\Herd\workopia\storage\app\public].

PS C:\Users\ellio\Herd\workopia>

php artisan storage:link

Now I can remove the dd (die and dump)

▼ app

▼ Http\ Controllers

Controller.php

HomeController.php

JobController.php

> Models

> Providers

> View

> bootstrap

> config

> database

> node_modules

> public

> resources

> routes

▼ storage

▼ app

> private

> public

◆ .gitignore

> framework

> OUTLINE

> TIMELINE

```
10 class JobController extends Controller
23     public function store(Request $request): RedirectResponse
42         'contact_phone' => 'nullable|string',
43         'company_name' => 'required|string',
44         'company_description' => 'nullable|string',
45         'company_logo' => 'nullable|image|mimes:jpeg,png,jpg,gif|max:2048',
46         'company_website' => 'nullable|url'
47     ];
48     // Temporarily Hardcode the user_id for new job inputs
49     $validatedData['user_id'] = 1;
50
51     // Check for image
52     if ($request->hasFile('company_logo')) {
53         // store file and get path
54         $path = $request->file('company_logo')->store('logos', 'public');
55     }
56
57     // Add path to validated company_logo
58     $validatedData['company_logo'] = $path;
59
60     // Submit to database
61     Job::create($validatedData);
62
63     return redirect()->route('jobs.index')->with('success', 'Job listing created successfully!');
64 }
65
```

And handle the image storage
in the storage\App\public
folder and get the symlink
path to put in the database.

EXPLORER

WORKOPIA

resources

views

pages

layout.blade.php

welcome.blade.php

routes

storage

app

private

public

logos

0fcQ9dLVgS6Ne5BwEfLMbRziobYf...

.gitignore

.gitignore

framework

logs

tests

vendor

JobController.php

alert.blade.php

0fcQ9dLVgS6Ne5BwEfLMbRziobYf1nDzxEhdgVpb.png

storage > app > public > logos > 0fcQ9dLVgS6Ne5BwEfLMbRziobYf1nDzxEhdgVpb.png

When I test the creat job form and select a logo, I see that it is getting put into the storage\App\public\logos folder with a random hash filename

And if I look in the database, the path to this file is being added to the company_logo column



| company_description
text | company_logo
character varying (255) |
|---|--|
| Vencom is a design-driven company focused on creating exceptional user experiences through innovative UX design. | logos/logo-vencom.png |
| Digital Media specializes in innovative online marketing strategies and digital content management. | logos/logo-digital-media.png |
| Pink Pig is a technology company dedicated to creating innovative products and solutions that drive industry advancement. | logos/logo-pink-pig.png |
| Tec Solutions provides comprehensive IT support and solutions, ensuring optimal performance of technology infrastructure. | logos/logo-tec-solutions.png |
| Aperiam temporibus f | logos/0fcQ9dLVgS6Ne5BwEfLMbRziobYf1nDzxEhdgVpb.png |

EXPLORER

WORKOPIA

resources

views

components

inputs

text.blade.php

textarea.blade.php

alert.blade.php

bottom-banner.blade.php

button-link.blade.php

header.blade.php

hero.blade.php

job-card.blade.php

nav-link.blade.php

topbanner.blade.php

jobs

pages

JobController.php

alert.blade.php

job-card.blade.php

resources > views > components > job-card.blade.php > article.rounded-lg.shadow-md.bg-white

```
1  @props(['job'])
2
3  <article class="rounded-lg shadow-md bg-white p-4">
4      <div class="flex items-center space-between gap-4">
5          @if($job->company_logo)
6              company_name }}"
9                  class="w-14"
10             />
11          @endif
12          <div>
13              <h2 class="text-xl font-semibold">{{ $job->title }}</h2>
14              <p class="text-sm text-gray-500">{{ $job->job_type }}</p>
15          </div>
16      </div>
17      <p class="text-gray-700 text-lg mt-2">{{ Str::limit($job->descript
18      <ul class="my-4 bg-gray-100 p-4 rounded">
19          <li class="mb-2">@strong>Salary: @strong>{{ number_format($fi
```

Update the job card blade so that the file path points to storage

I also need to move the logo images for the existing jobs from the public\images\logos to the storage\App\public\logos

EXPLORER

WORKOPIA

resources

views

components

alert.blade.php

bottom-banner.blade.php

button-link.blade.php

header.blade.php

hero.blade.php

job-card.blade.php

nav-link.blade.php

topbanner.blade.php

jobs

create.blade.php

index.blade.php

Show.blade.php

pages

home.blade.php

index.blade.php

layout.blade.php

welcome.blade.php

routes

OUTLINE

TIMELINE

JobController.php

alert.blade.php

job-card.blade.php

Show.blade.php

> jobs > Show.blade.php > x-layout > div.grid.grid-cols-1.md:grid-cols-4.gap-6 > aside.bg-white.round

1<x-layout>

2<div class="grid grid-cols-1 md:grid-cols-4 gap-6">

3<section class="md:col-span-3">

79</div>

80

81<div class="bg-white p-6 rounded-lg shadow-md mt-6">

82<div id="map"></div>

83</div>

84</section>

85

86<aside class="bg-white rounded-lg shadow-md p-3">

87<h3 class="text-xl text-center mb-4 font-bold">Company Info</h3>

88@if (\$job->company_logo)

89company_name }}"

92class="w-full rounded-lg mb-4 m-auto"

93/>

94@endif

95@if (\$job->company_name)

96<h4 class="text-lg font-bold">{{ \$job->company_name }}</h4>

97@endif

98@if (\$job->company_description)

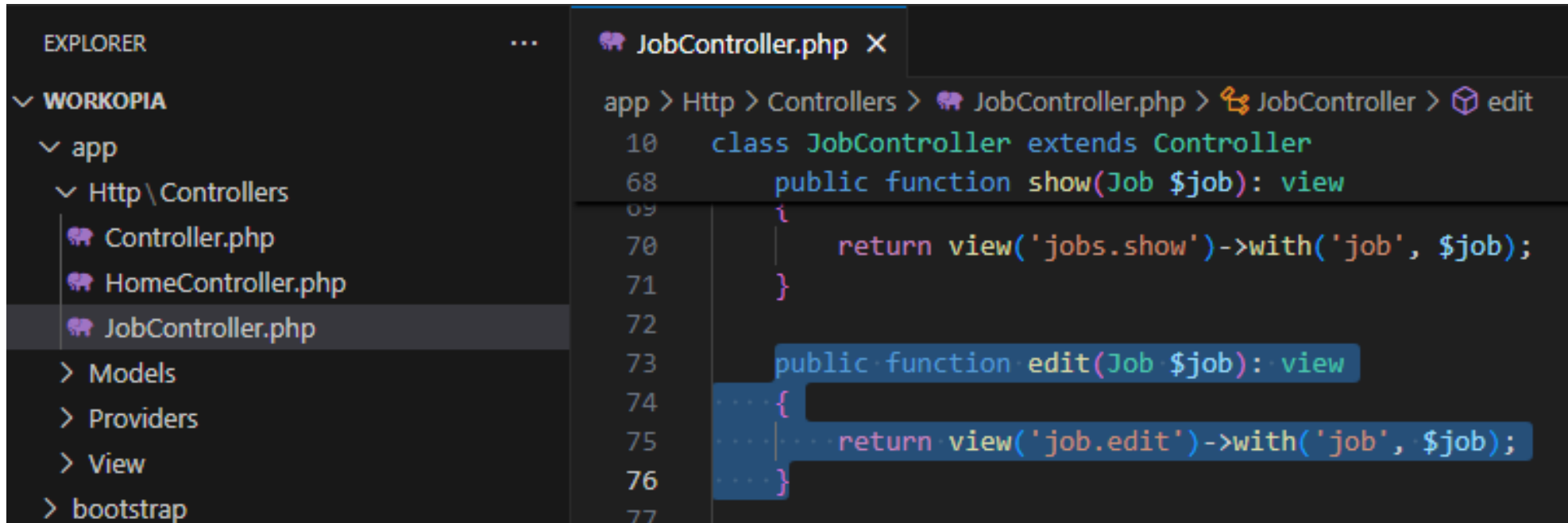
99<p class="text-gray-700 text-lg my-3">{{ \$job->company_description }}</p>

100@endif

101@if (\$job->company_website)

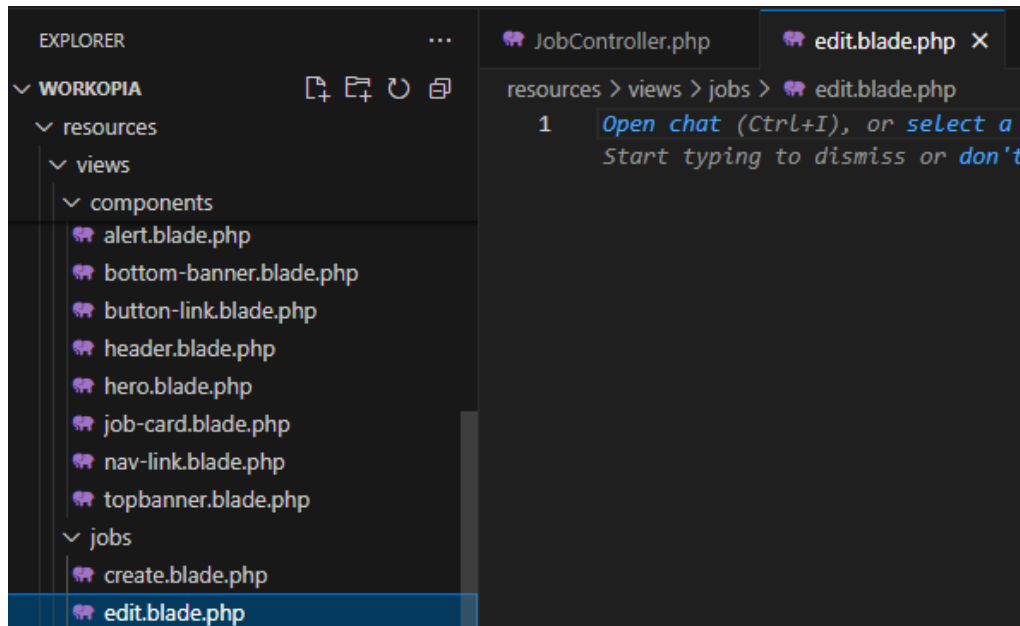
Update the job show blade also so that the file path points to storage

7.13 Update Job Listing



```
EXPLORER
WORKOPIA
  app
    Http\Controllers
      Controller.php
      HomeController.php
      JobController.php
    Models
    Providers
    View
    bootstrap

JobController.php X
app > Http > Controllers > JobController.php > JobController > edit
10  class JobController extends Controller
68      public function show(Job $job): view
69      {
70          return view('jobs.show')->with('job', $job);
71      }
72
73      public function edit(Job $job): view
74      {
75          return view('job.edit')->with('job', $job);
76      }
77
```



```
EXPLORER
WORKOPIA
  resources
    views
      components
        alert.blade.php
        bottom-banner.blade.php
        button-link.blade.php
        header.blade.php
        hero.blade.php
        job-card.blade.php
        nav-link.blade.php
        topbanner.blade.php
      jobs
        create.blade.php
        edit.blade.php

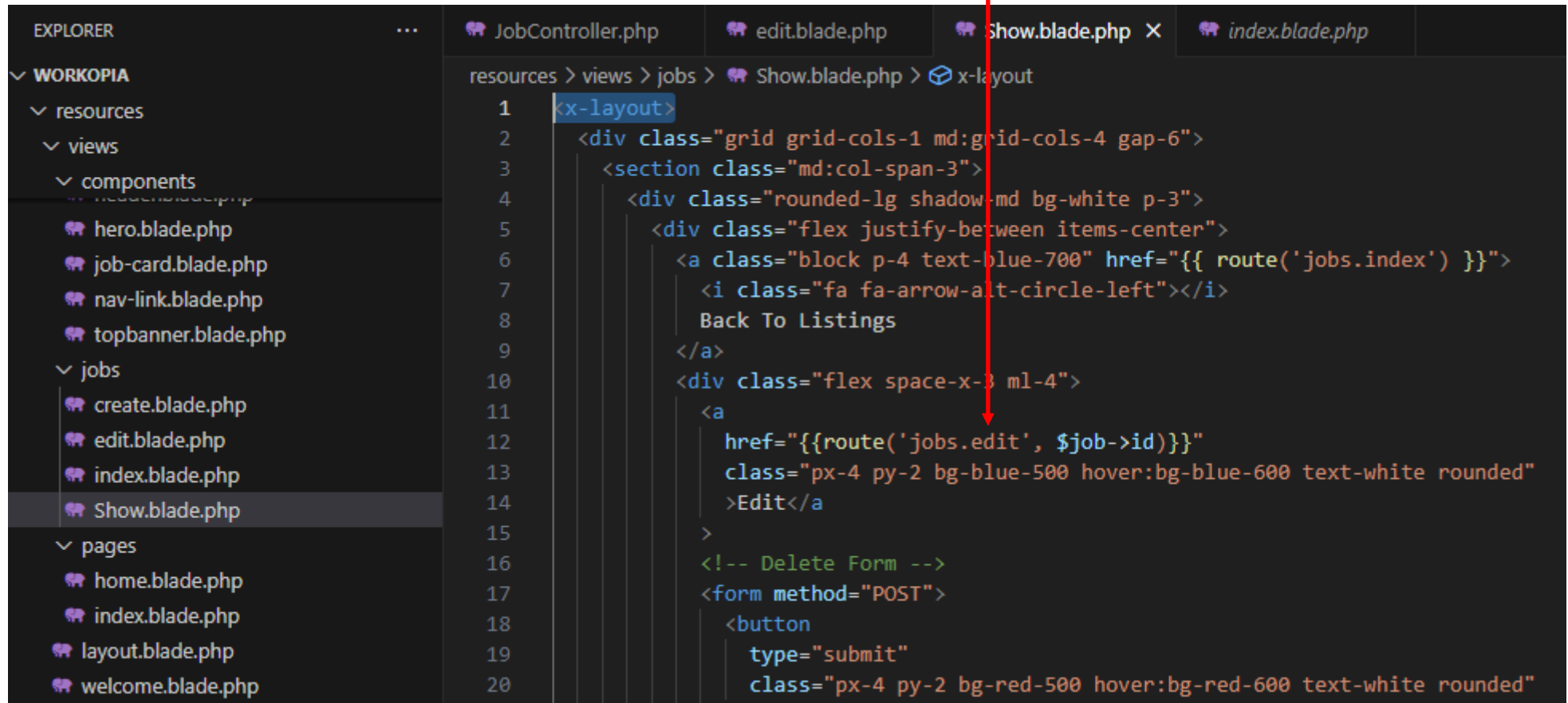
JobController.php
edit.blade.php X
resources > views > jobs > edit.blade.php
1  Open chat (Ctrl+I), or select a
   Start typing to dismiss or don't
```

In the Job controller, the edit function needs to return the job.edit view and pass in an object of the job.

I also need to create a new blade in views\jobs\ called edit.blade.php

In the show.blade I need to change the path so that the edit button points to the newly create edit.blade using the route helper and pass in the job Id using model binding.

`<a href="{{ route('jobs.edit', $job->id)}}">`



The screenshot shows the Visual Studio Code editor with the following components:

- EXPLORER:** A sidebar on the left showing the project structure. The 'resources' folder is expanded, showing 'views' and 'components'. The 'jobs' folder is also expanded, showing 'create.blade.php', 'edit.blade.php', 'index.blade.php', and 'Show.blade.php'.
- FILE EXPLORER:** A sidebar on the right showing the file structure. The 'resources' folder is expanded, showing 'views' and 'components'. The 'jobs' folder is also expanded, showing 'create.blade.php', 'edit.blade.php', 'index.blade.php', and 'Show.blade.php'.
- EDITOR:** The main workspace showing the 'Show.blade.php' file. The file is located at 'resources > views > jobs > Show.blade.php > x-layout'. The code is as follows:

```
1 <x-layout>
2 <div class="grid grid-cols-1 md:grid-cols-4 gap-6">
3   <section class="md:col-span-3">
4     <div class="rounded-lg shadow-md bg-white p-3">
5       <div class="flex justify-between items-center">
6         <a class="block p-4 text-blue-700" href="{{ route('jobs.index') }}">
7           <i class="fa fa-arrow-alt-circle-left"></i>
8           Back To Listings
9         </a>
10        <div class="flex space-x-3 ml-4">
11          <a
12            href="{{ route('jobs.edit', $job->id) }}"
13            class="px-4 py-2 bg-blue-500 hover:bg-blue-600 text-white rounded"
14            >Edit</a>
15          <!-- Delete Form -->
16          <form method="POST">
17            <button
18              type="submit"
19              class="px-4 py-2 bg-red-500 hover:bg-red-600 text-white rounded"
20            >
```

A red arrow points from the text above to the `href="{{ route('jobs.edit', $job->id) }}"` attribute in the code.

```

<x-layout>
  <div class="bg-white mx-auto p-8 rounded-lg shadow-md w-full md:max-w-3xl">
    <h2 class="text-4xl text-center font-bold mb-4">Edit Job Listing</h2>

    <!-- Form Start -->
    <form
      method="POST" action="{{ route('jobs.update', $job->id) }}" enctype="multipart/form-
data">
      @csrf @method('PUT')
      <h2 class="text-2xl font-bold mb-6 text-center text-gray-500">Job Info</h2>

      <!-- Job Title -->
      <x-inputs.text id="title" name="title" label="Job Title" placeholder="Software Engineer"
:value="old('title', $job->title)"/>

      <!-- Job Description -->
      <x-inputs.textarea id="description" name="description" label="Job Description"
placeholder="We are seeking a skilled and motivated Software Developer..."
:value="old('description', $job->description)"/>

      <!-- Annual Salary -->
      <x-inputs.text id="salary" name="salary" label="Annual Salary" type="number"
placeholder="90000" :value="old('salary', $job->salary)"/>

      <!-- Requirements -->
      <x-inputs.textarea id="requirements" name="requirements" label="Requirements"
placeholder="Bachelor's degree in Computer Science" :value="old('requirements', $job-
>requirements)"/>

      <!-- Benefits -->
      <x-inputs.textarea id="benefits" name="benefits" label="Benefits" placeholder="Health
insurance, 401k, paid time off" :value="old('benefits', $job->benefits)"/>

```

The edit blade is basically the same as the create blade

Use Laravel csrf method to make the request a PUT

Reuse input blade components for all the form fields

Values being passed in are either the old values or actual new value

```
<!-- Tags -->
<x-inputs.text id="tags" name="tags" label="Tags (comma-separated)"
placeholder="development,coding,java,python" :value="old('tags', $job->tags)"/>

<!-- Job Type -->
<x-inputs.select id="job_type" name="job_type" label="Job Type" :options="['Full-Time'
=> 'Full-Time', 'Part-Time' => 'Part-Time', 'Contract' => 'Contract', 'Temporary' =>
'Temporary', 'Internship' => 'Internship', 'Volunteer' => 'Volunteer', 'On-Call' => 'On-
Call']" :value="old('job_type', $job->job_type)"/>

<!-- Remote -->
<x-inputs.select id="remote" name="remote" label="Remote" :options="[0 => 'No', 1 =>
'Yes']" :value="old('remote', $job->remote)"/>

<h2 class="text-2xl font-bold mb-6 text-center text-gray-500">Company Info</h2>

<!-- Address -->
<x-inputs.text id="address" name="address" label="Address" placeholder="123 Main St"
:value="old('address', $job->address)"/>

<!-- City -->
<x-inputs.text id="city" name="city" label="City" placeholder="Albany"
:value="old('city', $job->city)"/>

<!-- State -->
<x-inputs.text id="state" name="state" label="State" placeholder="NY"
:value="old('state', $job->state)"/>

<!-- ZIP Code -->
<x-inputs.text id="zipcode" name="zipcode" label="ZIP Code" placeholder="12201"
:value="old('zipcode', $job->zipcode)"/>

<!-- Company Name -->
<x-inputs.text id="company_name" name="company_name" label="Company Name"
placeholder="Company name" :value="old('company_name', $job->company_name)"/>
```

```
<!-- Company Description -->
<x-inputs.textarea id="company_description" name="company_description" label="Company
Description" placeholder="Company Description" :value="old('company_description', $job-
>company_description)"/>

<!-- Company Website -->
<x-inputs.text id="company_website" name="company_website" label="Company Website"
type="url" placeholder="Enter website" :value="old('company_website', $job-
>company_website)"/>

<!-- Contact Phone -->
<x-inputs.text id="contact_phone" name="contact_phone" label="Contact Phone"
placeholder="Enter phone" :value="old('contact_phone', $job->contact_phone)"/>

<!-- Contact Email -->
<x-inputs.text id="contact_email" name="contact_email" label="Contact Email"
type="email" placeholder="Email where you want to receive applications"
:value="old('contact_email', $job->contact_email)"/>

<!-- Company Logo -->
<x-inputs.file id="company_logo" name="company_logo" label="Company Logo"/>

<!-- Submit Button -->
<button type="submit" class="w-full bg-green-500 hover:bg-green-600 text-white px-4 py-2
my-3 rounded focus:outline-none">Save</button>

</form>
</div>
</x-layout>
```

File Edit Selection View Go Run ...

EXPLORER

WORKOPIA

app

Http\ Controllers

Controller.php

HomeController.php

JobController.php

Models

Providers

View

bootstrap

config

JobController.php X edit.blade.php Show.blade.php

app > Http > Controllers > JobController.php > ...

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use Illuminate\View\View;
7 use App\Models\Job;
8 use Illuminate\Http\RedirectResponse;
9 use Illuminate\Support\Facades\Storage;
10
11 class JobController extends Controller
```

In the job controller, I need to bring in the storage facade


```
public function update(Request $request, Job $job): RedirectResponse
{
    // Validate the incoming request data
    $validatedData = $request->validate([
        'title' => 'required|string|max:255',
        'description' => 'required|string',
        'salary' => 'required|integer',
        'tags' => 'nullable|string',
        'job_type' => 'required|string',
        'remote' => 'required|boolean',
        'requirements' => 'nullable|string',
        'benefits' => 'nullable|string',
        'address' => 'nullable|string',
        'city' => 'required|string',
        'state' => 'required|string',
        'zipcode' => 'required|string',
        'contact_email' => 'required|email',
        'contact_phone' => 'nullable|string',
        'company_name' => 'required|string|max:255',
        'company_description' => 'nullable|string',
        'company_logo' => 'nullable|image|mimes:jpeg,png,jpg,gif|max:2048',
        'company_website' => 'nullable|url',
    ]);
}
```

In the update method of the 'jobs controller', the code is basically the same as the create method.

First the data is validated.

```
// Check if a file was uploaded
if ($request->hasFile('company_logo')) {
    // Delete the old company logo from storage
    if ($job->company_logo) {
        Storage::delete('public/logos/' . basename($job->company_logo));
    }
    // Store the file and get the path
    $path = $request->file('company_logo')->store('logos', 'public');

    // Add the path to the validated data array
    $validatedData['company_logo'] = $path;
}
```

If there is already a logo image then I want to delete it using the storage class.

And store the new image

```
// Update with the validated data
$job->update($validatedData);
```

And then on the instance of the job, I call the update method passing in the newly validated data

```
return redirect()->route('jobs.index')->with('success', 'Job listing updated successfully!');
}
```

Finally, the edit function ends with a redirect to the 'all jobs' page with a custom flash message

7.14 Delete Listing

```
<form method="POST" action="{{ route('jobs.destroy', $job->id) }}" onsubmit="return
confirm('Are you sure you want to delete this job?');">
    @csrf @method('DELETE')
    <button type="submit" class="px-4 py-2 bg-red-500 hover:bg-red-600 text-white
    rounded">Delete</button>
</form>
```

On the show blade, I want to modify the delete form so that it routes to the jobs controller destroy method, passing in the job->id. I also add a on submit message so that the user has to confirm the deletion. The CSRF Laravel method is delete.

```
public function destroy(Job $job): RedirectResponse
{
    // If there is a company logo, delete it from storage
    if ($job->company_logo) {
        Storage::delete('public/logos/' . $job->company_logo);
    }

    // Delete the job
    $job->delete();

    return redirect()->route('jobs.index')->with('success', 'Job listing deleted
successfully!');
}
```

In the jobs controller I handle the delete in the destroy method. The storage:delete Laravel method removes the image. Finally, I put in a redirect back to job listing page with message.

8. Authenticating & Creating Users

[8.1 Intro](#)

[8.2 Authentication Options](#)

[8.3 Laravel Breeze Setup](#)

[8.4 How Sessions Work
& Session Helpers](#)

[8.5 Login & Register
Control & Routes](#)

[8.6 Register New user](#)

[8.7 Login user](#)

[8.8 Logout and Auth Directive](#)

8.1 Intro



Laravel From Scratch

Authentication & Creating Users - Section Intro

- ✓ **Authentication Options**
- ✓ **Laravel Breeze Demo**
- ✓ **How Sessions Work**
- ✓ **Login & Register Controllers**
- ✓ **Register User Functionality**
- ✓ **Login Functionality**
- ✓ **Logout User**
- ✓ **@auth Directive**

8.2 Authentication Options

Authentication Options

- ✓ **Custom Authentication**
- ✓ **Laravel Breeze - Easy to use starter kit**
- ✓ **Laravel Jetstream - Feature-rich starter kit**
- ✓ **Laravel Fortify - Frontend agnostic Authentication**
- ✓ **Laravel Sanctum - API Token**
- ✓ **Socialite - Use OAuth with providers (Facebook, Google, etc)**

8.3 Laravel Breeze Setup

Breeze setup will install a bunch of new folders and forms into a project which might damage an existing project if it is already built so I have decided to create a new herd site called 'Elliott-ward-farmer-cv' where I will implement breeze.

```
PS C:\Users\ellio\Herd\elliott-ward-farmer-cv> composer require laravel/breeze
./composer.json has been updated
Running composer update laravel/breeze
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
  - Locking laravel/breeze (v2.3.7)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Downloading laravel/breeze (v2.3.7)
  - Installing laravel/breeze (v2.3.7): Extracting archive
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
```

Composer require Laravel/breeze

```
INFO Discovering packages.
```

| | |
|-----------------------------------|------|
| laravel/breeze | DONE |
| laravel/pail | DONE |
| laravel/sail | DONE |
| laravel/tinker | DONE |
| nesbot/carbon | DONE |
| nunomaduro/collision | DONE |
| nunomaduro/termwind | DONE |
| pestphp/pest-plugin-laravel | DONE |

```
86 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
> @php artisan vendor:publish --tag=laravel-assets --ansi --force
```

```
INFO No publishable resources for tag [laravel-assets].
```

```
No security vulnerability advisories found.
Using version ^2.3 for laravel/breeze
PS C:\Users\ellio\Herd\elliott-ward-farmer-cv>
```

```
PS C:\Users\ellio\Herd\elliott-ward-farmer-cv> php artisan breeze:install
```

Php artisan breeze:install

```
Which Breeze stack would you like to install?
```

```
Blade with Alpine ..... blade
Livewire (Volt Class API) with Alpine ..... livewire
Livewire (Volt Functional API) with Alpine ..... livewire-functional
React with Inertia ..... react
Vue with Inertia ..... vue
API only ..... api
```

```
> blade
```

Select front end framework

```
Would you like dark mode support? (yes/no) [no]
```

```
> yes
```

Dark mode support? Y or N

```
Which testing framework do you prefer? [Pest]
```

```
Pest ..... 0
PHPUnit ..... 1
```

```
>
```

Testing framework, I left in blank

```
./composer.json has been updated
```

```
Running composer update pestphp/pest pestphp/pest-plugin-laravel
```

```
Loading composer repositories with package information
```

```
Updating dependencies
```

```
Nothing to modify in lock file
```

```
Writing lock file
```

```
Installing dependencies from lock file (including require-dev)
```

```
Nothing to install, update or remove
```

```
Generating optimized autoload files
```

```
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
```

```
> @php artisan package:discover --ansi
```

```
INFO Discovering packages.
```



```
laravel/breeze ..... DONE
laravel/pail ..... DONE
laravel/sail ..... DONE
laravel/tinker ..... DONE
nesbot/carbon ..... DONE
nunomaduro/collision ..... DONE
nunomaduro/termwind ..... DONE
pestphp/pest-plugin-laravel ..... DONE
```

86 packages you are using are looking for funding.

Use the `composer fund` command to find out more!

```
> @php artisan vendor:publish --tag=laravel-assets --ansi --force
```

```
INFO No publishable resources for tag [laravel-assets].
```

No security vulnerability advisories found.

Using version ^3.8 for pestphp/pest

Using version ^3.2 for pestphp/pest-plugin-laravel

```
INFO Installing and building Node dependencies.
```

added 104 packages, changed 1 package, and audited 196 packages in 25s

49 packages are looking for funding

run `npm fund` for details

found 0 vulnerabilities

```
> build
```

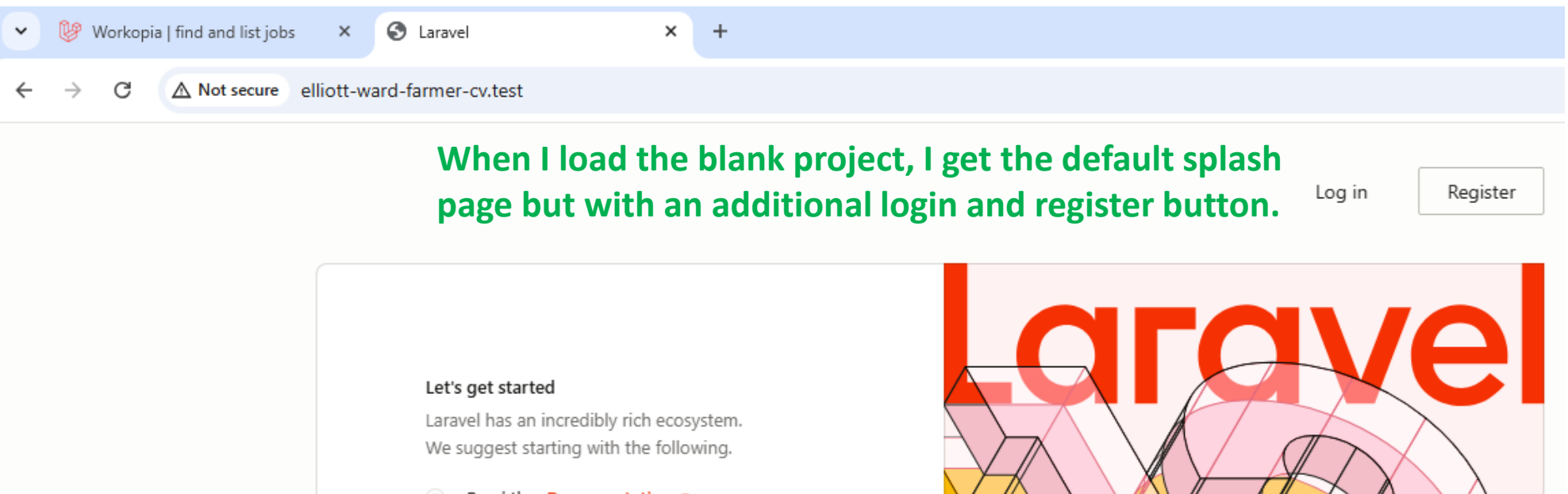
```
> vite build
```

```
vite v6.3.5 building for production...
transforming...
  ✓ 54 modules transformed.
rendering chunks...
computing gzip size...
public/build/manifest.json      0.27 kB | gzip: 0.15 kB
public/build/assets/app-DbTWgVAB.css 38.35 kB | gzip: 6.95 kB
public/build/assets/app-DaBYqt0m.js 79.84 kB | gzip: 29.77 kB
✓ built in 18.34s
```

INFO Breeze scaffolding installed successfully.

```
PS C:\Users\ellio\Herd\elliott-ward-farmer-cv>
```

And away it goes installing all the dependencies, views, classes, models etc for breeze to work.



When I load the blank project, I get the default splash page but with an additional login and register button.

Without writing a line of code I
have a register page.



Name

Elliott Farmer

Email

elliottbcn@gmail.com

Password

.....

Confirm Password

.....

[Already registered?](#)

REGISTER

Profile

Profile Information

Update your account's profile information and email address.

Name

Elliott Farmer

Email

elliottbcn@gmail.com

SAVE

It also comes with a user dashboard page where they can update their profile, change password etc...

Activate Windows
Go to Settings to activate Windows.



Type here to search



29°C Soleado



ENG
ES

17:57
24/06/2025



8.4 How Sessions work & Session Helpers

What Are Sessions?

A session is a way to store information across multiple requests in your application.

HTTP is a stateless protocol. Each request is independent of the previous request.

Sessions are used to maintain state information, such as user authentication status, flash messages, etc

Authentication Process

- ✓ User logs in and creates a session on the server.
- ✓ When we submit a job listing, we want to get the user id from the session and store with the listing.
- ✓ A cookie is stored on the client with the session ID
- ✓ The cookie is sent to the server with every request
- ✓ There are also “remember me” cookies

WORKOPIA

app

Http\ Controllers

HomeController.php

JobController.php

Models

Providers

View

bootstrap

config

app.php

auth.php

cache.php

database.php

filesystems.php

logging.php

mail.php

queue.php

services.php

session.php

database

JobController.php

session.php

edit.blade.php

Show.blade.php

create.blade.php

config > session.php

```

1 <?php
2
3 use Illuminate\Support\Str;
4
5 return [
6
7     /*
8      |-----
9      | Default Session Driver
10     |-----
11     |
12     | This option determines the default session driver that is utilized for
13     | incoming requests. Laravel supports a variety of storage options to
14     | persist session data. Database storage is a great default choice.
15     |
16     | Supported: "file", "cookie", "database", "memcached",
17     |             "redis", "dynamodb", "array"
18     |
19     */
20
21     'driver' => env('SESSION_DRIVER', 'database'),
22

```

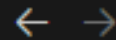
In the config folder of the Laravel project there is a php file called session that handles all the sessions

In the database there is a table called 'sessions' that logs the sessions. It has an ID, user ID, ip address, user agent, payload (encrypted) and last activity.

| | id
[PK] character varying (255) | user_id
bigint | ip_address
character var | user_agent
text | payload
text | last_activity
integer |
|---|---|-------------------|-----------------------------|--|------------------------|--------------------------|
| 1 | LaKf0yFlvfVHvkd9G9tNnKM4ZE6nR8s31O71R... | [null] | 127.0.0.1 | Mozilla/5.0 (Windows NT 10.0; Win64; x64) A... | YTozOntzOjY6II90b2t... | 1750784717 |
| 2 | MKCGyJUUsQdFVE7gBhioEv52t93cAvIERJc5cO... | [null] | 127.0.0.1 | Mozilla/5.0 (Windows NT 10.0; Win64; x64) A... | YTozOntzOjY6II90b2t... | 1750782837 |
| 3 | wFHGq9jJtCu0T9sf7yhkrp0xvTxgvGfA3vZnWa7z | [null] | 127.0.0.1 | Mozilla/5.0 (Windows NT 10.0; Win64; x64) A... | YTozOntzOjY6II90b2t... | 1750772001 |



File Edit Selection View Go Run ...



workopia



EXPLORER



WORKOPIA

app

Http\ Controllers

Controller.php

HomeController.php

JobController.php

Models

Providers

View

bootstrap

config

app.php

auth.php

cache.php

database.php

filesystems.php

logging.php

mail.php

JobController.php

HomeController.php X

edit.blade.php

app > Http > Controllers > HomeController.php > HomeController > index.php

```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6  use Illuminate\View\View;
7  use App\Models\Job;
8
9  class HomeController extends Controller
10 {
11     public function index(): View
12     {
13         If I go to the controller and add a line of
14         code that says: Session()->put('test', '123');
15         session()->put('test', '123');
16
17         $jobs = Job::latest()->limit(6)->get();
18
19         return view('pages.home')->with('jobs', $jobs);
20     }
21 }
```


| | id
[PK] character varying (255) | user_id
bigint | ip_address
character varying (45) | user_agent
text | payload
text | last_activity
integer |
|---|--|-------------------|--------------------------------------|----------------------|----------------------|--------------------------|
| 1 | LaKf0yFlvfVHvkd9G9tNnKM4ZE6nR8s31O71R... | [null] | 127.0.0.1 | Mozilla/5.0 (Wind... | YTozOntzOjY6II90b... | 1750784717 |
| 2 | MKCGyJUsQdFVE7gBhioEv52t93cAvIERJc5cO... | [null] | 127.0.0.1 | Mozilla/5.0 (Wind... | YTozOntzOjY6II90b... | 1750782837 |
| 3 | VY8aqDNwYTJ5xcPtH7CnKRbySnFDojgmQmh... | [null] | 127.0.0.1 | Mozilla/5.0 (Wind... | YTo0OntzOjY6II90b... | 1750785511 |
| 4 | wFHGq9jJtCu0T9sf7yhkrp0xvTxgvGfA3vZnWa7z | [null] | 127.0.0.1 | Mozilla/5.0 (Wind... | YTozOntzOjY6II90b... | 1750772001 |

After reloading the homepage, I can see an extra session in the database. I cannot see the payload of 123 because it is encrypted but I can get the payload in another controller.

EXPLORER

WORKOPIA

app

Http\Controllers

Controller.php

HomeController.php

JobController.php

Models

JobController.php

app > Http > Controllers > JobController.php > JobController > create

```

11 class JobController extends Controller
12
13     public function index(): View
14     {
15         $value = session()->get('test');
16         dd($value);
17     }

```

workopia.test/jobs

workopia.test/jobs

"123" // app\Http\Controllers\JobController.php:16

I can get the session value

In jobs controller, I can place the session value of 'test' into a variable and then die dump

WORKOPIA

app

Http\Controllers

Controller.php

HomeController.php

JobController.php

Models

Providers

View

bootstrap

config

app.php

app > Http > Controllers > HomeController.php > ...
9 class HomeController extends Controller
10 {
11 public function index(): View
12 {
13
14 session()->put('test', '123');
15 session()->forget('test');
16
17 \$jobs = Job::latest()->limit(6)->get();
18
19 return view('pages.home')->with('jobs', \$jobs);
20 }
21 }

I can forget the session

workopia.test/jobs

workopia.test/jobs

null // app\Http\Controllers\JobController.php:16

So when I go back to the all jobs page then return back to the home page the session is now Null.

8.5 Login & Register Controllers & Routes

```
PS C:\Users\ellio\Herd\workopia> php artisan make:controller LoginController
```

```
INFO Controller [C:\Users\ellio\Herd\workopia\app\Http\Controllers\LoginController.php]
created successfully.
```

```
PS C:\Users\ellio\Herd\workopia> php artisan make:controller RegisterController
```

```
INFO Controller
[C:\Users\ellio\Herd\workopia\app\Http\Controllers\RegisterController.php] created
successfully.
```

```
PS C:\Users\ellio\Herd\workopia>
```

```
<?php
```

```
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\JobController;
use App\Http\Controllers\HomeController;

use App\Http\Controllers\LoginController;
use App\Http\Controllers\RegisterController;
```

```
Route::get('/', [HomeController::class, 'index'])->name('home');
Route::resource('jobs', JobController::class);
```

```
Route::get('/register', [RegisterController::class, 'register'])->name('register');
Route::post('/register', [RegisterController::class, 'store'])->name('register.store');
Route::get('/login', [LoginController::class, 'login'])->name('login');
Route::post('/login', [LoginController::class, 'authenticate'])->name('login/authenticate');
```

I am going to create some additional named routes for the register and login functionality.

```

<?php

namespace App\Http\Controllers;

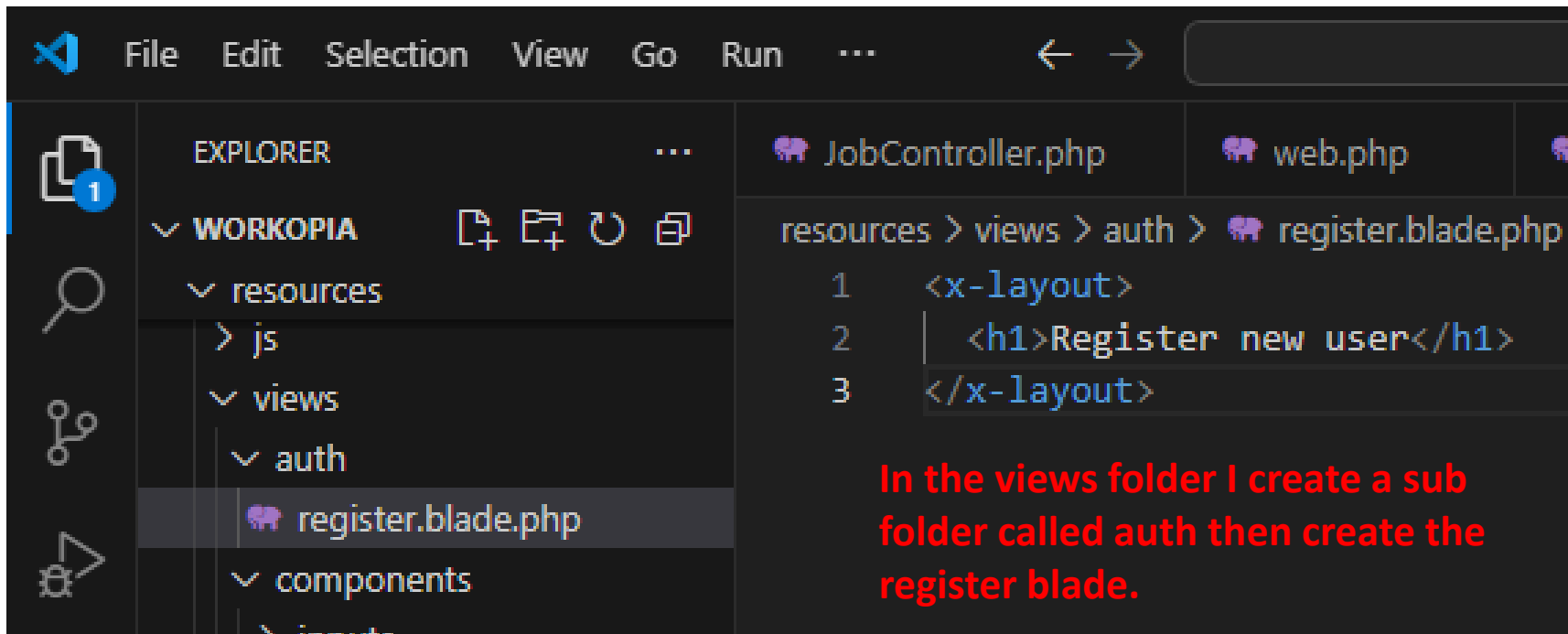
use Illuminate\Http\Request;
use Illuminate\View\View;
use Illuminate\Http\RedirectResponse;

class RegisterController extends Controller
{
    public function register(): View
    {
        return view('auth.register');
    }
}

```

I add the needed view and RedirectResponse classes to the register controller.

I create the public function for register and type cast it to a view. The function returns a view authenticate user blade.



In the views folder I create a sub folder called auth then create the register blade.

The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a project structure with a folder named 'WORKOPIA' containing an 'app' directory. Inside 'app', there is an 'Http\Controllers' folder where 'LoginController.php' is highlighted. The main editor area shows the code for 'LoginController.php' with the following content:

```
1  <?php
2
3  namespace App\Http\Controllers;
4
5  use Illuminate\Http\Request;
6  use Illuminate\View\View;
7  use Illuminate\Http\RedirectResponse;
8
9  class LoginController extends Controller
10 {
11     public function login(): View
12     {
13         return view('auth.login');
14     }
15 }
```

Copy and paste
to make the login
controller with a
login function
that returns a
view of
auth.login

The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays a project structure with a folder named 'WORKOPIA' containing a 'resources' directory. Inside 'resources', there is a 'views' folder, and within it, an 'auth' folder where 'login.blade.php' is highlighted. The main editor area shows the code for 'login.blade.php' with the following content:

```
1  <x-layout>
2  <h1>Login user</h1>
3  </x-layout>
```

Copy and paste
to make the login
blade view

8.6 Register New User

```
<x-layout>
  <div class="bg-white rounded-lg shadow-md w-full md:max-w-xl mx-auto mt-12 p-8 py-12">
    <h2 class="text-4xl text-center font-bold mb-4">Register</h2>
    <form method="POST" action="{{ route('register.store') }}">
      @csrf

      <x-inputs.text id="name" name="name" placeholder="Full name" />

      <x-inputs.text id="email" name="email" type="email" placeholder="Email address"/>

      <x-inputs.text id="password" name="password" type="password" placeholder="Password"/>

      <x-inputs.text id="password_confirmation" name="password_confirmation" type="password"
placeholder="Confirm Password"/>

      <button type="submit" class="w-full bg-blue-500 hover:bg-blue-600 text-white px-4 py-2
rounded focus:outline-none">Register</button>

      <p class="mt-4 text-gray-500">Already have an account?<a class="text-blue-900"
href="{{route('login')}}">Login</a></p>

    </form>
  </div>
</x-layout>
```

**The register form reuses the input components.
Note the form action goes to a store function.**

```

<?php
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use Illuminate\View\View;
use Illuminate\Http\RedirectResponse;
use Illuminate\Support\Facades\Hash;
use App\Models\User;

class RegisterController extends Controller
{
    // @desc Register new user
    // @route GET /register
    public function register(): view
    {
        return view('auth.register');
    }

    // @desc Store User in the Database
    // @route POST /register
    public function store(Request $request): RedirectResponse
    {
        // Validate the incoming request data
        $validatedData = $request->validate([
            'name' => 'required|string|max:255',
            'email' => 'required|string|email|max:255|unique:users',
            'password' => 'required|string|min:8|confirmed',
        ]);
        // hash the password
        $validatedData['password'] = Hash::make($validatedData['password']);
        // Add the user to the database
        $user = User::create($validatedData);
        // redirect to the login with success message
        return redirect()->route('login')->with('success', 'Registration successful You can now log
in!');
    }
}

```

I need to bring in the hash class which provides a helper function called `bcrypt`. I also need the user class to interact with the user table of the database.

The store function is similar in functionality to the create job. Data is validated before passing it to the database.

Hash::make will hash the user password

Now when use the submit on the register form, I see that a new user has been added to the database.

| id
[PK] bigint | name
character varying (255) | email
character varying (255) | email_verified_at
timestamp without time zone | password
character varying (255) | remember_token
character varying (255) |
|-------------------|---------------------------------|----------------------------------|--|---|---|
| 7 | Maximus Schmidt | metz.wilhelm@example.net | 2025-06-22 20:12:06 | \$2y\$12\$Ljg5paVNUfZkv3hZXB8IFO/i0CM2vNwV2alwGoudsSeTxK0x... | 5wjlgZa... |
| 8 | Silas Abshire | wsauer@example.com | 2025-06-22 20:12:06 | \$2y\$12\$Ljg5paVNUfZkv3hZXB8IFO/i0CM2vNwV2alwGoudsSeTxK0x... | PMNpiCE... |
| 9 | Prof. Domenica Lynch | littel.jovan@example.net | 2025-06-22 20:12:06 | \$2y\$12\$Ljg5paVNUfZkv3hZXB8IFO/i0CM2vNwV2alwGoudsSeTxK0x... | 7ABgMoC... |
| 10 | Leland Johns | uschmeler@example.org | 2025-06-22 20:12:06 | \$2y\$12\$Ljg5paVNUfZkv3hZXB8IFO/i0CM2vNwV2alwGoudsSeTxK0x... | FS6IE0GS... |
| 11 | Elliott Farmer | elliottbcn@gmail.com | [null] | \$2y\$12\$XYqukkhUVtA.ndoRlin4v.RU9IoEKyZfd2rZwb9DAWftm4iLhlvH2 | [null] |

8.7 Login User

```
<x-layout>
  <div
    class="bg-white rounded-lg shadow-md w-full md:max-w-xl mx-auto mt-12 p-8 py-12"
  >
    <h2 class="text-4xl text-center font-bold mb-4">Login</h2>

    <form method="POST" action="{{ route('login.authenticate') }}"
      @csrf

        <x-inputs.text id="email" name="email" type="email" placeholder="Email address"/>

        <x-inputs.text id="password" name="password" type="password" placeholder="Password"/>

        <button type="submit" class="w-full bg-blue-500 hover:bg-blue-600 text-white px-4 py-2
rounded focus:outline-none">Login</button>

        <p class="mt-4 text-gray-500">Don't have an account?<a class="text-blue-900"
href="{{route('register')}}">Register</a></p>

      </form>
    </div>
</x-layout>
```

The login form reuses the input blade components

```
<?php
```

```
namespace App\Http\Controllers;
```

```
use Illuminate\Http\Request;
```

```
use Illuminate\View\View;
```

```
use Illuminate\Http\RedirectResponse;
```

```
use Illuminate\Support\Facades\Auth;
```

```
class LoginController extends Controller
```

```
{
```

```
    // @desc Login user
```

```
    // @route GET /login
```

```
    public function login(): view
```

```
    {
```

```
        return view('auth.login');
```

```
    }
```

I need to bring in an auth class which has all of the Laravel methods for authenticating a user

```

// @desc Authenticate user
// @route GET /login/authenticate
public function authenticate(Request $request): RedirectResponse
{
    // Validate the request data
    $credentials = $request->validate([
        'email' => 'required|email',
        'password' => 'required|string',
    ]);

    // Attempt to log the user in
    if (Auth::attempt($credentials)) {
        // Regenerate the session to prevent fixation attacks
        $request->session()->regenerate();
        // Redirect to the intended route or a default route
        return redirect()->intended(route('home'))->with('success', 'You are now logged
in!');
    }
    // If authentication fails, redirect back with an error message
    return back()->withErrors([
        'email' => 'The provided credentials do not match our records.',
    ])->onlyInput('email');
}
}

```

As before I validate the inputs

Now the auth::attempt method will check the email and password against those in the database

I need to also handle an authentication failure with a generic error message

Now if I login I have a session cookie called `Laravel_session` in the browser

The screenshot shows a web browser at `workopia.test` displaying the Workopia website. The website has a blue header with the text "Workopia" and a search bar. The main content area has the text "Find Your Dream Job" and "Unlock Your Career Potential". The Chrome DevTools Application tab is open, showing the "Cookies" section for the domain `https://workopia.test`. The cookies listed are:

| Name | Value | Domain | Path | Expires | Size | HttpOnly | Secure | SameSite | Priority |
|-----------------|----------------------------|--------|------|---------|------|----------|--------|----------|----------|
| laravel_session | eyJpdil6lkWUGIiLOV0eS9q... | wo... | / | 20... | 357 | ✓ | ✓ | Lax | M... |
| XSRF-TOKEN | eyJpdil6ljRRMGRndVk2OX... | wo... | / | 20... | 352 | | ✓ | Lax | M... |

A red arrow points from the text "Now if I login I have a session cookie called `Laravel_session` in the browser" to the `laravel_session` cookie in the table.

| | id
[PK] character varying (255) | user_id
bigint | ip_address
character varying (45) | user_agent
text |
|---|--|-------------------|--------------------------------------|---|
| 1 | EBzTiPjJ9yCxqR8ir92TIHmOQKmdHd71ZPFCd... | 11 | 127.0.0.1 | Mozilla/5.0 (Windows NT 10.0; Win64; x64) A |
| 2 | LaKf0yFlvfVHvkd9G9tNnKM4ZE6nR8s31O71RLID | [null] | 127.0.0.1 | Mozilla/5.0 (Windows NT 10.0; Win64; x64) A |

The session is also logged in the database

8.8 Logout User

I am adding a route for the logout. It goes to the login controller. I could make a sperate controller for logout but it is not really needed.

```
Route::post('/logout', [LoginController::class, 'logout'])->name('logout');
```

The logout method in the controller uses the Auth::logout method. As well as killing the session (invalidate) and it also regenerates the CSRF token, finally redirecting back to the homepage.

```
// @desc Logout user
// @route POST /logout
public function logout(Request $request): RedirectResponse
{
    Auth::logout(); // Log out the user

    $request->session()->invalidate(); // Invalidate the session
    $request->session()->regenerateToken(); // Regenerate the CSRF token

    return redirect('/');
}
```

In the header blade I can use the `@auth` directive to chose which buttons show when a user is logged in. for example when logged-in I don't want to see the login or register links. I repeat the same logic for the mobile menu

```
<nav class="hidden md:flex items-center space-x-4">

  <x-nav-link url="/" :active="request()->is('/')">Home</x-nav-link>
  <x-nav-link url="/jobs" :active="request()->is('jobs')">All Jobs</x-nav-link>
  @auth
  <x-nav-link url="/jobs/saved" :active="request()->is('jobs/saved')">Saved Jobs</x-nav-link>
  <x-nav-link url="/dashboard" :active="request()->is('profile')" icon="gauge"> Dashboard</x-nav-link>
  <x-button-link url="/jobs/create" icon="edit">Create Job</x-button-link>
  @else
  <x-nav-link url="/login" :active="request()->is('login')">Login</x-nav-link>
  <x-nav-link url="/register" :active="request()->is('register')">Register</x-nav-link>
  @endAuth

</nav>
```

```
<form action="{{route'logout'}}"
method="post">
  @csrf
  <button type="submit" class="text-
white">
    <i class="fa fa-sign-out"></i>
  Logout
  </button>
</form>
```

I create a `logout.blade` that I can then add to the header blade, placing the `<x-logout-button>` between the create job and the dashboard.

9. Middleware, Authorisation & policies

[9.1 Intro](#)

[9.2 Middleware Overview](#)

[9.3 Protecting Routes](#)

[9.4 Guest Middleware](#)

[9.5 Test User Seeder](#)

[9.6 Add Current user to Listing](#)

[9.7 Policies & @can Directive](#)

[9.8 Policy Authorisation in
Controller](#)

9.1 Intro



Laravel From Scratch

Middleware, Authorization & Policies - Section Intro

- ✓ **Middleware Overview**
- ✓ **Protecting Routes**
- ✓ **Route Middleware Groups**
- ✓ **Login & Register Controllers**
- ✓ **Test User Seeder**
- ✓ **Policies & Authorization**
- ✓ **@can Directive**
- ✓ **Job Ownership**

9.2 Middleware Overview

What Is Middleware?

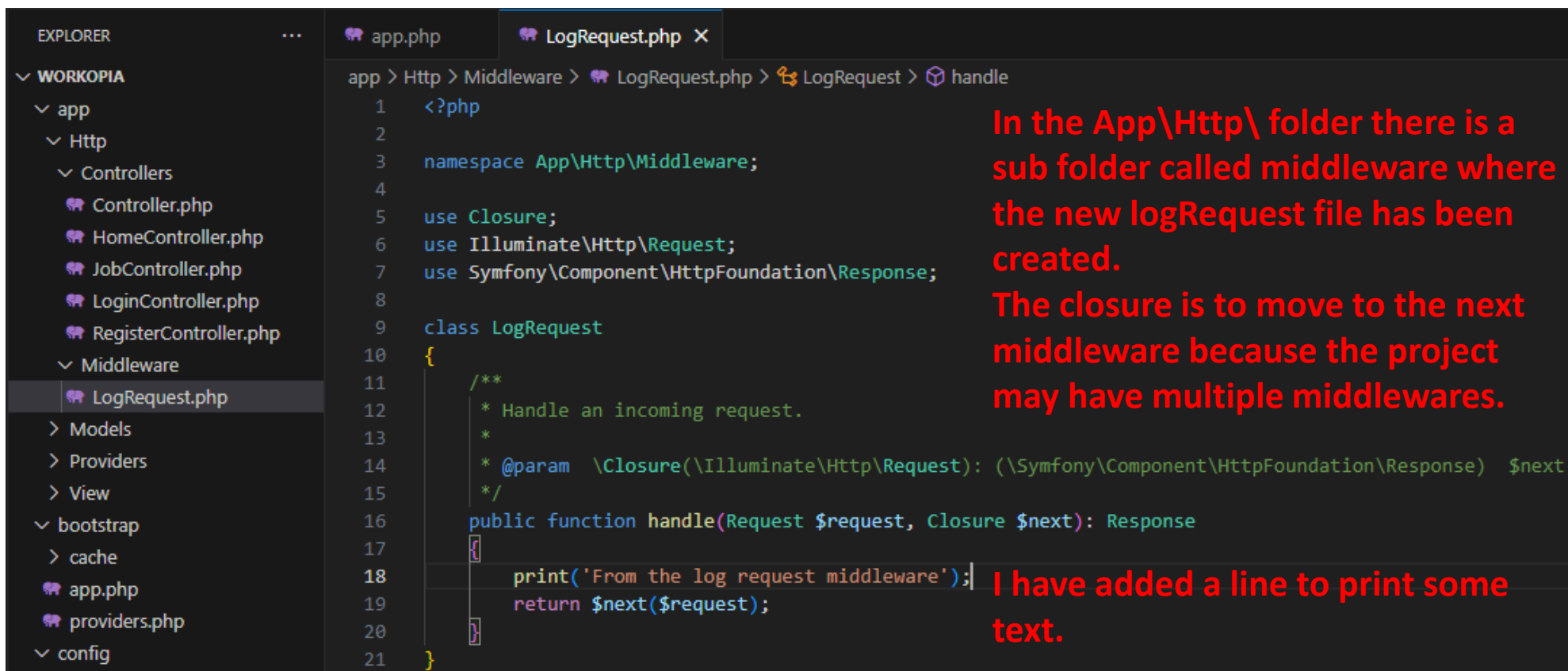
Middleware is a type of filter that sits between the HTTP request from the client and the application. It checks and processes incoming requests before they reach the controller or outgoing responses before they are sent back to the user.

There is global middleware as well as route-based middleware

```
PS C:\Users\ellio\Herd\workopia> php artisan make:middleware LogRequest
```

INFO Middleware [C:\Users\ellio\Herd\workopia\app\Http\Middleware\LogRequest.php] created successfully.

```
PS C:\Users\ellio\Herd\workopia>
```



EXPLOLER

WORKOPIA

- app
 - Http
 - Controllers
 - Controller.php
 - HomeController.php
 - JobController.php
 - LoginController.php
 - RegisterController.php
 - Middleware
 - LogRequest.php**
 - Models
 - Providers
 - View
 - bootstrap
 - cache
 - app.php
 - providers.php
 - config

app > Http > Middleware > LogRequest.php > LogRequest > handle

```
1 <?php
2
3 namespace App\Http\Middleware;
4
5 use Closure;
6 use Illuminate\Http\Request;
7 use Symfony\Component\HttpFoundation\Response;
8
9 class LogRequest
10 {
11     /**
12      * Handle an incoming request.
13      *
14      * @param \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response) $next
15      */
16     public function handle(Request $request, Closure $next): Response
17     {
18         print('From the log request middleware');
19         return $next($request);
20     }
21 }
```

In the App\Http\ folder there is a sub folder called middleware where the new logRequest file has been created.

The closure is to move to the next middleware because the project may have multiple middlewares.

I have added a line to print some text.

EXPLORER

WORKOPIA

app

Http

Controllers

RegisterController.php

Middleware

LogRequest.php

Models

Providers

View

bootstrap

cache

app.php

providers.php

config

app.php

auth.php

cache.php

database.php

filesystems.php

logging.php

outline

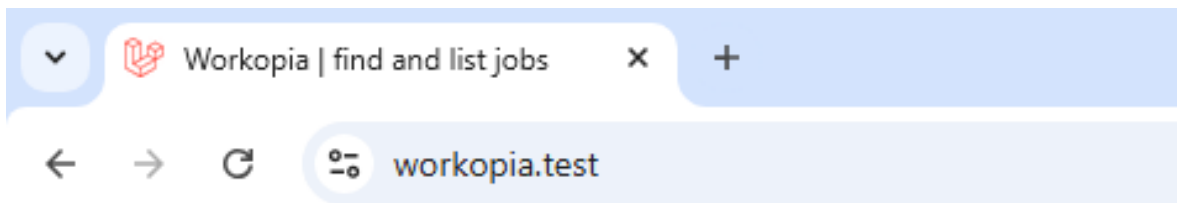
app.php

LogRequest.php

vite.config.js

bootstrap > app.php > Closure

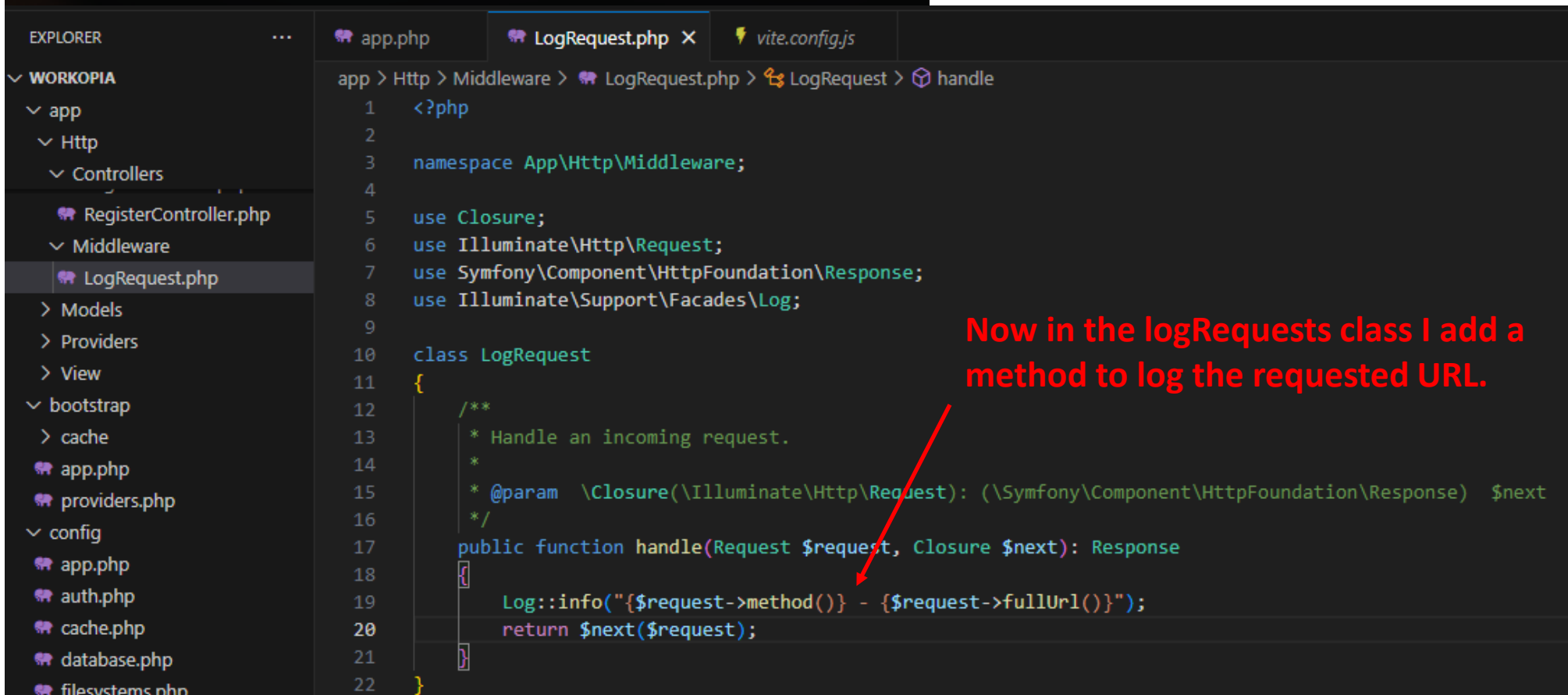
```
1 <?php
2
3 use Illuminate\Foundation\Application;
4 use Illuminate\Foundation\Configuration\Exceptions;
5 use Illuminate\Foundation\Configuration\Middleware;
6 use App\Http\Middleware\LogRequest; I bring in the LogRequest class
7
8 return Application::configure(basePath: dirname(__DIR__))
9     ->withRouting(
10         web: __DIR__ . '/../routes/web.php',
11         commands: __DIR__ . '/../routes/console.php',
12         health: '/up',
13     )
14     ->withMiddleware(function (Middleware $middleware): void {
15         $middleware->validateCsrfTokens(except: [
16             '/submit'
17         ]);
18
19         $middleware->append(logRequest::class); And append it to the middleware
20     })
21     ->withExceptions(function (Exceptions $exceptions): void {
22         //
23     }->create());
24
```



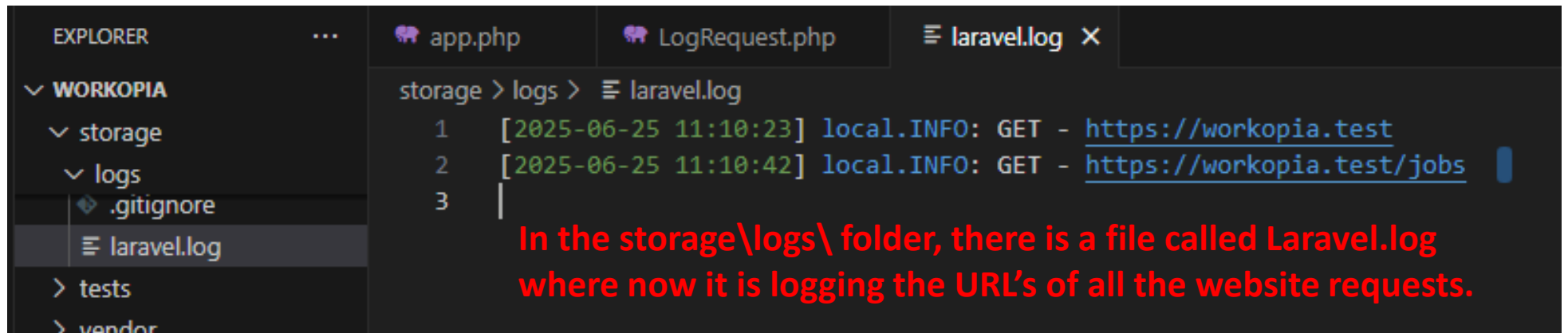
Whichever page I am on, I get the print text on the screen

From the log request middleware

Workopia



Now in the logRequests class I add a method to log the requested URL.



But I don't want to use this globally, only for certain routes so in the Bootstrap\App.php file I will comment out the `// $middleware->append(logRequest::class);`;



9.3 Protect Routes

Even though I am not logged in, and the create job is missing, I can still manually append jobs/create to the base URL and access the create jobs method. This is undesirable and there are multiple ways of preventing this. The first method is from the job controller.

```
use Illuminate\Support\Facades\Auth;
...

public function create(): View | RedirectResponse
{
    if (!auth::check()) {
        return redirect()->route('login');
    }

    return view('jobs.create');
}
```

In the jobs controller I bring in the auth class

Now I can have an if statement that checks if the user is NOT authenticated and redirects to the login page

The downside of this method is that it will get very cumbersome having to add a check login to every method of every controller where I want to restrict access for non logged in users.

The elegant and slick way of performing route protection is using the routes/web.php file

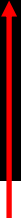
```
<?php
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\JobController;
use App\Http\Controllers\HomeController;
use App\Http\Controllers\LoginController;
use App\Http\Controllers\RegisterController;

Route::get('/', [HomeController::class, 'index'])->name('home');
// Route::resource('jobs', JobController::class);
Route::resource('jobs', JobController::class)->middleware('auth')->only(['create', 'edit', 'update', 'destroy']);
Route::resource('jobs', JobController::class)->except(['create', 'edit', 'update', 'destroy']);
Route::get('/register', [RegisterController::class, 'register'])->name('register');
Route::post('/register', [RegisterController::class, 'store'])->name('register.store');
Route::get('/login', [LoginController::class, 'login'])->name('login');
Route::post('/login', [LoginController::class, 'authenticate'])->name('login.authenticate');
Route::post('/logout', [LoginController::class, 'logout'])->name('logout');
```

I want to use auth middleware to protect only the create, edit, update and destroy methods of jobs



I also have to add a line after to allow all resource routes except create, edit, update and destroy



← → ↻ workopia.test/jobs/2

Workopia

Home All Jobs Login Register

← Back To Listings

Edit Delete

Marketing Specialist

Bitwave is seeking a creative and strategic Marketing Specialist to develop and execute comprehensive marketing campaigns. In this role, you will be responsible for market research, identifying target audiences, and crafting compelling messages to drive brand awareness and engagement. You'll

Company Info



I am logged out and I can view all jobs. I can also view job details (job.show) but if I click on the delete button or the edit button (jobs.edit) it takes me to the login page because these methods are now protected in the route file.

Equally if I type jobs/create in the browser url, it takes me to the login page because this route is also protected.

9.4 Guest Middleware

Guest does the opposite of the auth. So, if I want to protect routes from being accessed when logged in then I can use guest.

```
<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\JobController;
use App\Http\Controllers\HomeController;
use App\Http\Controllers\LoginController;
use App\Http\Controllers\RegisterController;

Route::get('/', [HomeController::class, 'index'])->name('home');
// Route::resource('jobs', JobController::class);
Route::resource('jobs', JobController::class)->middleware('auth')->only(['create', 'edit', 'update', 'destroy']);
Route::resource('jobs', JobController::class)->except(['create', 'edit', 'update', 'destroy']);

Route::middleware('guest')->group(function () {
    Route::get('/register', [RegisterController::class, 'register'])->name('register');
    Route::post('/register', [RegisterController::class, 'store'])->name('register.store');
    Route::get('/login', [LoginController::class, 'login'])->name('login');
    Route::post('/login', [LoginController::class, 'authenticate'])->
>name('login.authenticate');
});

Route::post('/logout', [LoginController::class, 'logout'])->name('logout');
```

If I am logged in then I should not be able to access any login or register routes. I can do this by grouping these routes inside of the guest middleware function.

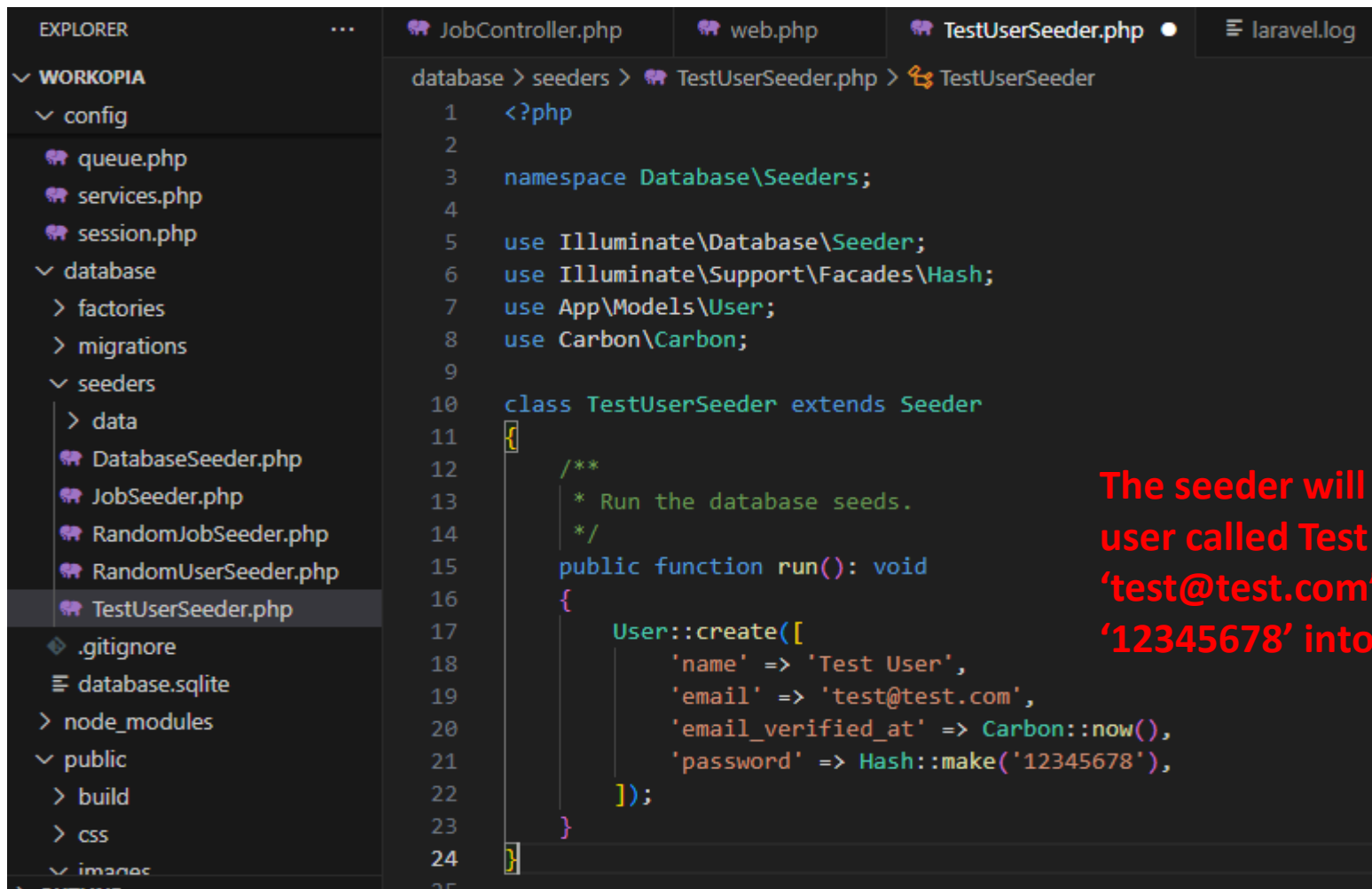
If I try to add /login or /register in the browser then it does not take me to these pages when I am logged in

9.5 Test User Seeder

```
PS C:\Users\ellio\Herd\workopia> php artisan make:seeder TestUserSeeder
```

INFO Seeder [C:\Users\ellio\Herd\workopia\database\seeders\TestUserSeeder.php] created successfully.

```
PS C:\Users\ellio\Herd\workopia> Creating a test user seeder for the purposes of dev testing
```



```
EXPLORER
WORKOPIA
  config
  queue.php
  services.php
  session.php
  database
    factories
    migrations
    seeders
      data
      DatabaseSeeder.php
      JobSeeder.php
      RandomJobSeeder.php
      RandomUserSeeder.php
      TestUserSeeder.php
  .gitignore
  database.sqlite
  node_modules
  public
    build
    css
    images
  OUTLINE

JobController.php
web.php
TestUserSeeder.php
laravel.log

database > seeders > TestUserSeeder.php > TestUserSeeder
1  <?php
2
3  namespace Database\Seeders;
4
5  use Illuminate\Database\Seeder;
6  use Illuminate\Support\Facades\Hash;
7  use App\Models\User;
8  use Carbon\Carbon;
9
10 class TestUserSeeder extends Seeder
11 {
12     /**
13      * Run the database seeds.
14      */
15     public function run(): void
16     {
17         User::create([
18             'name' => 'Test User',
19             'email' => 'test@test.com',
20             'email_verified_at' => Carbon::now(),
21             'password' => Hash::make('12345678'),
22         ]);
23     }
24 }
```

The seeder will inject a now user called Test User with email 'test@test.com' and password '12345678' into the database.

EXPLORER

WORKOPIA

config

queue.php

services.php

session.php

database

factories

migrations

seeders

data

DatabaseSeeder.php

JobSeeder.php

RandomJobSeeder.php

RandomUserSeeder.php

TestUserSeeder.php

.gitignore

database.sqlite

node_modules

public

JobController.php

web.php

TestUserSeeder.php

DatabaseSeeder.php

laravel...

database > seeders > DatabaseSeeder.php > DatabaseSeeder > run

```
6 // use Illuminate\Database\Console\Seeds\WithoutModelEvents;
7 use Illuminate\Database\Seeder;
8 use Illuminate\Support\Facades\DB;
9
10 class DatabaseSeeder extends Seeder
11 {
12     /**
13      * Seed the application's database.
14      */
15     public function run(): void
16     {
17         // Truncate tables
18         DB::table('job_listings')->truncate();
19         DB::table('users')->truncate();
20
21         $this->call(TestUserSeeder::class); // <-- Test seeder added to db seeder
22         $this->call(RandomUserSeeder::class);
23         $this->call(JobSeeder::class);
24     }
25 }
26
```

The seeder is added to the database seeder

EXPLORER

WORKOPIA

config

queue.php

services.php

session.php

database

factories

migrations

seeders

data

DatabaseSeeder.php

JobSeeder.php

RandomJobSeeder.php

RandomUserSeeder.php

TestUserSeeder.php

.gitignore

database.sqlite

node_modules

public

build

css

images

OUTLINE

TIMELINE

JobController.php

web.php

TestUserSeeder.php

DatabaseSeeder.php

JobSeeder.php X

database > seeders > JobSeeder.php > ...

10

class JobSeeder extends Seeder

12

public function run(): void

13

{

14

// Load job listings data

15

\$jobListings = include database_path('seeders/data/job_listings.php');

16

17

// Get the ID of the user created by TestUserSeeder

18

\$testUserId = User::where('email', 'test@test.com')->value('id');

19

20

// Get all other user IDs

21

\$userIds = User::where('email', '!=', 'test@test.com')->pluck('id')->toArray();

22

23

foreach (\$jobListings as \$index => &\$listing) {

24

if (\$index < 2) {

25

// Assign the first two job listings to the test user

26

\$listing['user_id'] = \$testUserId;

27

} else {

28

// Assign the rest to random users

29

\$listing['user_id'] = \$userIds[array_rand(\$userIds)];

30

}

31

// Add timestamps

32

\$listing['created_at'] = now();

33

\$listing['updated_at'] = now();

34

}

35

36

// Insert job listings

The job seeder is modified so that the first two jobs in the database are owned by the test user

```
PS C:\Users\ellio\Herd\workopia> php artisan db:seed
```

And reseed the database

```
INFO Seeding database.
```

```
Database\Seeders\TestUserSeeder ..... RUNNING
Database\Seeders\TestUserSeeder ..... 467 ms DONE

Database\Seeders\RandomUserSeeder ..... RUNNING
Users Created Succesfully Database\Seeders\RandomUserSeeder ..... 1,188 ms DONE

Database\Seeders\JobSeeder ..... RUNNING
Database\Seeders\JobSeeder ..... 88 ms DONE
```

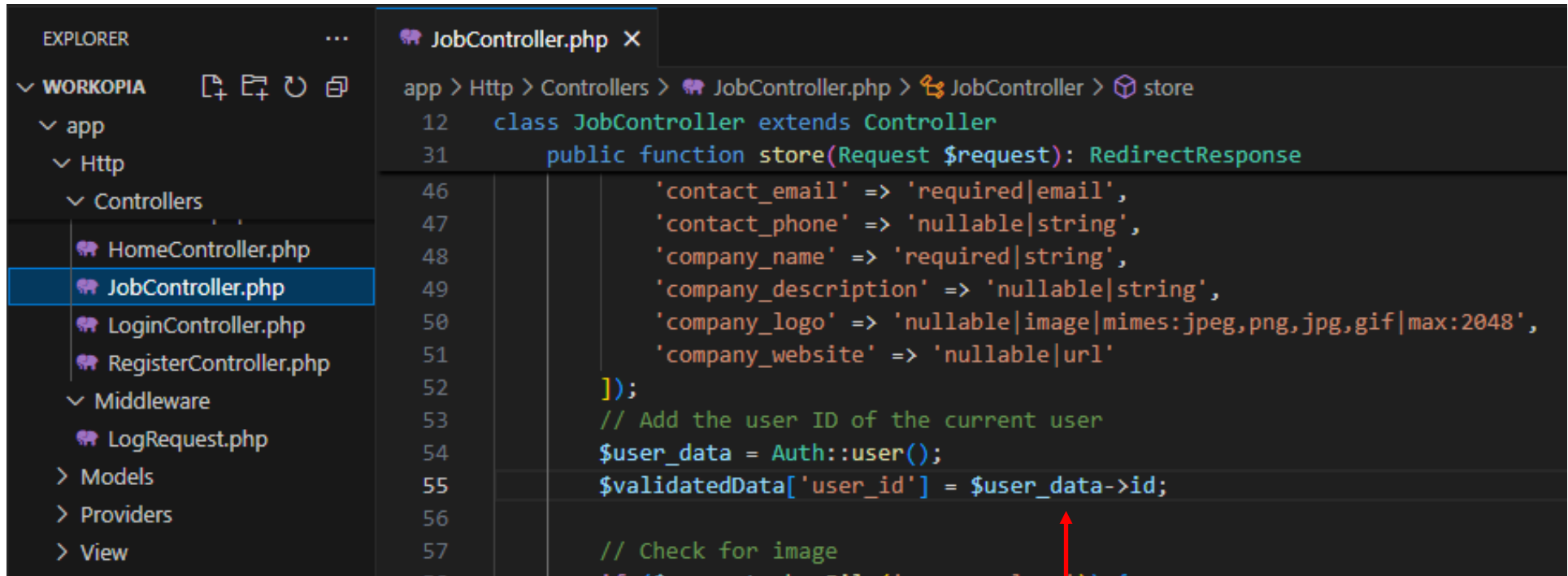
```
PS C:\Users\ellio\Herd\workopia>
```

Now, in the database jobs listings table, the first two jobs are owned by the test user

| | id
[PK] bigint | created_at
timestamp without time zone | updated_at
timestamp without time zone | user_id
bigint | title
character varying (255) | description
text |
|----|-------------------|---|---|-------------------|----------------------------------|--|
| 1 | 1 | 2025-06-25 13:00:01 | 2025-06-25 13:00:01 | 1 | Software Engineer | As a Software Engineer at Algorix, you will be responsible for de |
| 2 | 2 | 2025-06-25 13:00:01 | 2025-06-25 13:00:01 | 1 | Marketing Specialist | Bitwave is seeking a creative and strategic Marketing Specialist |
| 3 | 3 | 2025-06-25 13:00:01 | 2025-06-25 13:00:01 | 8 | Web Developer | At NextGen Solutions, our Web Developer will be crucial in build |
| 4 | 4 | 2025-06-25 13:00:01 | 2025-06-25 13:00:01 | 9 | Data Analyst | Quantum Code is in search of a Data Analyst to join our team ar |
| 5 | 5 | 2025-06-25 13:00:01 | 2025-06-25 13:00:01 | 10 | Graphic Designer | As a Graphic Designer at Shield, you will be at the forefront of o |
| 6 | 6 | 2025-06-25 13:00:01 | 2025-06-25 13:00:01 | 6 | Data Scientist | Join Sparkle as a Data Scientist and play a pivotal role in analyz |
| 7 | 7 | 2025-06-25 13:00:01 | 2025-06-25 13:00:01 | 5 | UX Designer | Vencom is seeking a UX Designer to enhance our user experienc |
| 8 | 8 | 2025-06-25 13:00:01 | 2025-06-25 13:00:01 | 8 | Digital Media Specialist | Digital Media is seeking a Digital Media Specialist to manage ar |
| 9 | 9 | 2025-06-25 13:00:01 | 2025-06-25 13:00:01 | 10 | Product Manager | Pink Pig is searching for a Product Manager to lead the develop |
| 10 | 10 | 2025-06-25 13:00:01 | 2025-06-25 13:00:01 | 5 | IT Support Specialist | Tec Solutions is hiring an IT Support Specialist to provide techni |

9.6 Add Current user to Job Listing

I have full authentication implemented in the application. However, I still need to implement authorization. Right now, anyone can edit or delete any listing. I need to implement authorization so that only the user who created a listing can edit or delete it.



```
EXPLORER
WORKOPIA
  app
    Http
      Controllers
        HomeController.php
        JobController.php
        LoginController.php
        RegisterController.php
    Middleware
      LogRequest.php
    Models
    Providers
    View

JobController.php X
app > Http > Controllers > JobController.php > JobController > store
12 class JobController extends Controller
31 public function store(Request $request): RedirectResponse
46     'contact_email' => 'required|email',
47     'contact_phone' => 'nullable|string',
48     'company_name' => 'required|string',
49     'company_description' => 'nullable|string',
50     'company_logo' => 'nullable|image|mimes:jpeg,png,jpg,gif|max:2048',
51     'company_website' => 'nullable|url'
52 ];
53 // Add the user ID of the current user
54 $user_data = Auth::user();
55 $validatedData['user_id'] = $user_data->id;
56
57 // Check for image
58 if ($request->hasFile('company_logo')) {
59     $image = $request->file('company_logo');
60     $image->move(public_path('uploads'), $image->getClientOriginalName());
61     $image->delete();
62 }
```

In the job controller store method, I want to now change it to storing a new job with default user of 1 to storing a new job with the current authenticated user id

EXPLORER

WORKOPIA

public

resources

css

js

views

auth

components

jobs

create.blade.php

edit.blade.php

index.blade.php

Show.blade.php

pages

home.blade.php

index.blade.php

layout.blade.php

welcome.blade.php

routes

storage

tests

JobController.php 1

Show.blade.php X

resources > views > jobs > Show.blade.php > x-layout > div.grid.grid-cols-1.md:grid-cols-4.gap-6 > section.md:col-span-3 > div.rounded-lg.shadow-
1 <x-layout>
2 <div class="grid grid-cols-1 md:grid-cols-4 gap-6">
3 <section class="md:col-span-3">
4 <div class="rounded-lg shadow-md bg-white p-3">
5 <div class="flex justify-between items-center">
6
7
8 Back To Listings
9
10 @auth @if (auth()->user()->id === \$job->user_id)
11 <div class="flex space-x-3 ml-4">
12 id) }}" class="px-4 py-2 bg-blue-500 hover:bg-blue-600
text-white rounded">Edit
13 <!-- Delete Form -->
14 <form method="POST" action="{{ route('jobs.destroy', \$job->id) }}" onsubmit="return confirm('Are you
sure you want to delete this job?');">
15 @csrf @method('DELETE')
16 <button type="submit" class="px-4 py-2 bg-red-500 hover:bg-red-600 text-white rounded">Delete</button>
17 </form>
18 <!-- End Delete Form -->
19 </div>
20 @endif @endauth
21 </div>
22 <div class="p-4">
23 <h2 class="text-xl font-semibold">{{ \$job->title }}</h2>

I use the @auth and @if directives for the edit and delete button div to check if the job user_id is the same as the logged in user id.

Now the edit and delete buttons will only show if the job is owned by the logged in user

← → ↻ workopia.test/jobs/3

Workopia Home All Jobs Saved Jobs Dashboard Logout Create Job

[Back To Listings](#)

Web Developer

At NextGen Solutions, our Web Developer will be crucial in building and maintaining exceptional web applications that delight users and meet business objectives. You will be involved in the full development lifecycle, including design, coding, testing, and deployment. The role requires expertise in front-end technologies such as HTML, CSS, and JavaScript, as

Company Info



NextGen Solutions

When logged in as test user and viewing job 3 the edit and delete buttons are not available

← → ↻ workopia.test/jobs/1

Workopia Home All Jobs Saved Jobs Dashboard Logout Create Job

[Back To Listings](#)

Software Engineer

As a Software Engineer at Algorix, you will be responsible for designing,

[Edit](#) [Delete](#)

Company Info



When logged in as test user and viewing job 1 the edit and delete buttons are available

workopia.test/jobs/1

Workopia

HomeAll JobsLoginRegister

If logged out then the edit and delete buttons should not be available for any job listing

⬅️ Back To Listings

Software Engineer

As a Software Engineer at Algorix, you will be responsible for designing, developing, and maintaining high-quality software applications. You will work closely with cross-functional teams to deliver scalable and efficient solutions that meet business needs. The role involves writing clean, maintainable code, participating in code reviews, and staying current with

Company Info



Algorix

9.7 Policies & @can Directive

Even though the buttons are not visible on the job listing show page, a logged in user still has access to the edit form via the browser url for a job that the user does not own. We can solve this using policies and the @directive.

```
PS C:\Users\ellio\Herd\workopia> php artisan make:policy JobPolicy --model=Job
```

```
INFO Policy [C:\Users\ellio\Herd\workopia\app\Policies\JobPolicy.php] created successfully.
```

```
PS C:\Users\ellio\Herd\workopia>
```

```
EXPLORER
WORKOPIA
  app
    Http
      Controllers
        Controller.php
        HomeController.php
        JobController.php
        LoginController.php
        RegisterController.php
    Middleware
    Models
    Policies
      JobPolicy.php
    Providers
    View
    bootstrap
    config
```

```
JobController.php JobPolicy.php X Show.blade.php
app > Policies > JobPolicy.php > JobPolicy > delete
9 class JobPolicy
34
35 /**
36  * Determine whether the user can update the model.
37  */
38 public function update(User $user, Job $job): bool
39 {
40     return $user->id == $job->user_id;
41 }
42
43 /**
44  * Determine whether the user can delete the model.
45  */
46 public function delete(User $user, Job $job): bool
47 {
48     return $user->id == $job->user_id;
49 }
50
```

In Http\Policies\ this creates a file called JobPolicy which contains all the default methods.

For the update method I want to return a boolean of the check if user id is the same as the job user_id

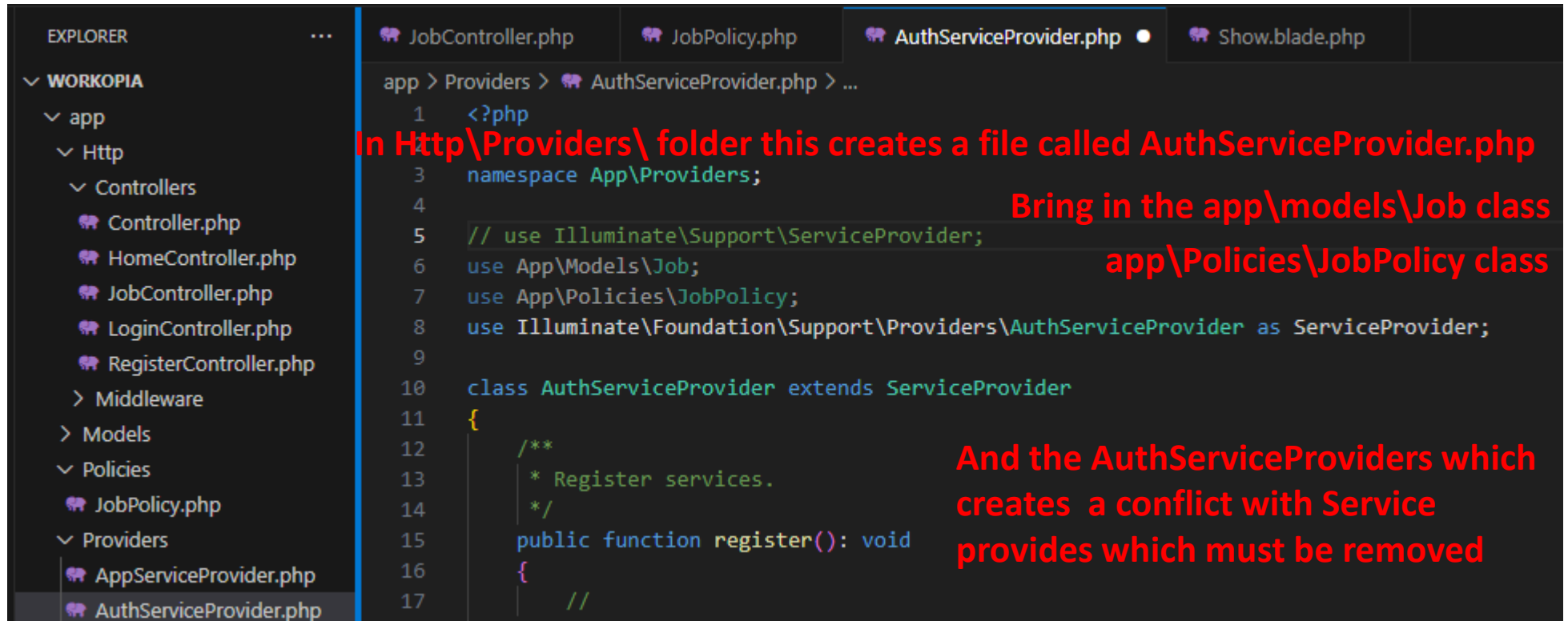
Same check for the delete method

I need to register this policy within an auth service provider. by create a new auth service provider with the following command:

```
PS C:\Users\ellio\Herd\workopia> php artisan make:provider AuthServiceProvider
```

```
INFO Provider [C:\Users\ellio\Herd\workopia\app\Providers\AuthServiceProvider.php] created successfully.
```

```
PS C:\Users\ellio\Herd\workopia>
```



The screenshot shows an IDE with the Explorer pane on the left and the Editor pane on the right. The Explorer pane shows the project structure with folders like WORKOPIA, app, Http, Controllers, Models, Policies, and Providers. The Editor pane shows the contents of the file `AuthServiceProvider.php` located at `app > Providers > AuthServiceProvider.php`. The file content is as follows:

```
1 <?php
2
3 namespace App\Providers;
4
5 // use Illuminate\Support\ServiceProvider;
6 use App\Models\Job;
7 use App\Policies\JobPolicy;
8 use Illuminate\Foundation\Support\Providers\AuthServiceProvider as ServiceProvider;
9
10 class AuthServiceProvider extends ServiceProvider
11 {
12     /**
13      * Register services.
14      */
15     public function register(): void
16     {
17         //
18     }
19 }
```

Red annotations are overlaid on the image:

- In Http\Providers\ folder this creates a file called AuthServiceProvider.php** (pointing to the file path in the Explorer pane)
- Bring in the app\models\Job class** (pointing to line 6)
- app\Policies\JobPolicy class** (pointing to line 7)
- And the AuthServiceProviders which creates a conflict with Service provides which must be removed** (pointing to the class definition on line 10)

When you generate a service provider in Laravel the default class that gets extended is that ``Illuminate\Support\ServiceProvider``. This is because service providers are typically meant to be used to register services, bindings, etc within the service container, which is the primary purpose of the `ServiceProvider` class.

However, for certain types of service providers, like `AuthServiceProvider`, Laravel provides specialized base classes such as this ``Illuminate\Foundation\Support\Providers\AuthServiceProvider`` that include additional functionality specific to that domain, such as the ``registerPolicies`` method, which is what we need to call. That method is not available in the original ``Illuminate\Support\ServiceProvider`` class.

WORKOPIA

app

Http

Controllers

Controller.php

HomeController.php

JobController.php

LoginController.php

RegisterController.php

Middleware

Models

Policies

JobPolicy.php

Providers

AppServiceProvider.php

AuthServiceProvider.php

View

bootstrap

config

database

app > Providers > AuthServiceProvider.php > AuthServiceProvider > boot

```
10 class AuthServiceProvider extends ServiceProvider
11 {
12     protected $policies = [
13         Job::class => JobPolicy::class,
14     ];
15     /**
16      * Register services.
17      */
18     public function register(): void
19     {
20         //
21     }
22
23     /**
24      * Bootstrap services.
25      */
26     public function boot(): void
27     {
28         $this->registerPolicies();
29     }
30 }
```

I need to create a policies property and connect the Job Model to the job policy

In the boot function I need to tell it to load the registerPolicies

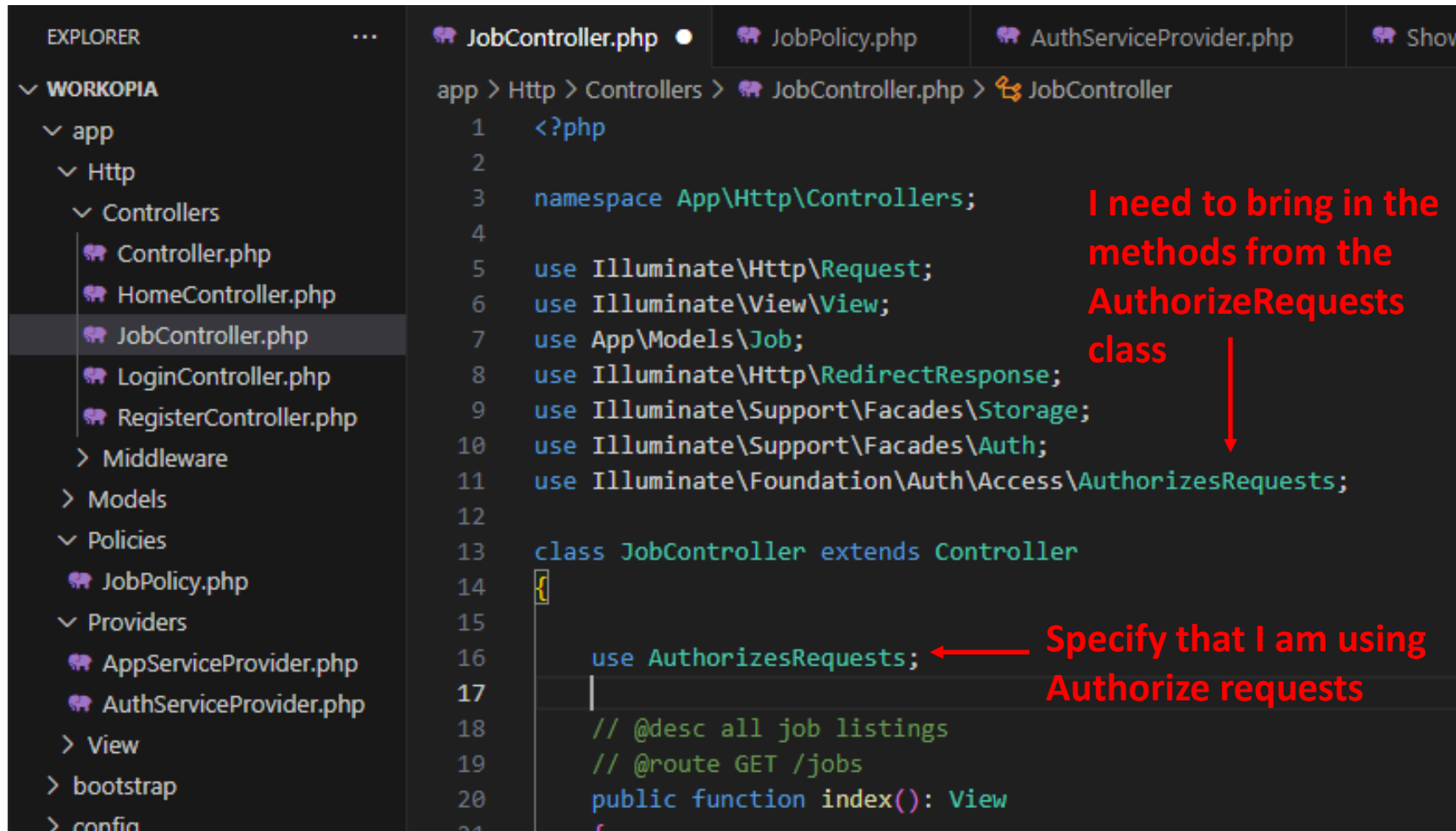
```
1 <x-layout>
2 <div class="grid grid-cols-1 md:grid-cols-4 gap-6">
3   <section class="md:col-span-3">
4     <div class="rounded-lg shadow-md bg-white p-3">
5       <div class="flex justify-between items-center">
6         <a class="block p-4 text-blue-700 href="{{ route('jobs.index') }}">
7           <i class="fa fa-arrow-alt-circle-left"></i>
8           Back To Listings
9         </a>
10        @can('update', $job)
11          <div class="flex space-x-3 ml-4">
12            <a href="{{ route('jobs.edit', $job->id) }}" class="px-4 py-2 bg-blue-500 hover:bg-blue-600
13              text-white rounded">Edit</a>
14            <!-- Delete Form -->
15            <form method="POST" action="{{ route('jobs.destroy', $job->id) }}" onsubmit="return confirm('Are you
16              sure you want to delete this job?');">
17              @csrf @method('DELETE')
18              <button type="submit" class="px-4 py-2 bg-red-500 hover:bg-red-600 text-white rounded">Delete</button>
19            </form>
20            <!-- End Delete Form -->
21          </div>
22        @endcan
23      </div>
24      <div class="p-4">
```

When it test it in the browser it works as before with the edit and delete buttons only showing for jobs that the test user owns. If I change the job id in the browser to a job that the test user does not own then the edit and delete button are not showing.

For logged out users the edit and delete buttons doe not show for any job show

9.8 Policy Authorisation in Controller

Right now, the policy is only preventing the user from seeing the edit and delete buttons. I need to use the policy in the job controller so they actually can't update or delete unless they own the listing.



The screenshot shows an IDE with the Explorer pane on the left and the Code editor on the right. The Explorer pane shows the project structure with the following folders and files:

- WORKOPIA
 - app
 - Http
 - Controllers
 - Controller.php
 - HomeController.php
 - JobController.php**
 - LoginController.php
 - RegisterController.php
 - Middleware
 - Models
 - Policies
 - JobPolicy.php
 - Providers
 - AppServiceProvider.php
 - AuthServiceProvider.php
 - View
 - bootstrap
 - config

The Code editor shows the contents of JobController.php:

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use Illuminate\View\View;
7 use App\Models\Job;
8 use Illuminate\Http\RedirectResponse;
9 use Illuminate\Support\Facades\Storage;
10 use Illuminate\Support\Facades\Auth;
11 use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
12
13 class JobController extends Controller
14 {
15
16     use AuthorizesRequests;
17
18     // @desc all job listings
19     // @route GET /jobs
20     public function index(): View
21 {
```

Two red annotations are present:

- A red arrow points from the text "I need to bring in the methods from the AuthorizeRequests class" to the `use Illuminate\Foundation\Auth\Access\AuthorizesRequests;` line (line 11).
- A red arrow points from the text "Specify that I am using Authorize requests" to the `use AuthorizesRequests;` line (line 16).

EXPLORER

WORKOPIA

app

Http

Controllers

Controller.php

HomeController.php

JobController.php

LoginController.php

RegisterController.php

Middleware

Models

Policies

JobPolicy.php

Providers

AppServiceProvider.php

AuthServiceProvider.php

View

JobController.php

JobPolicy.php

AuthServiceProvider.php

Show.blade.php

app > Http > Controllers > JobController.php > JobController > update

13

class JobController extends Controller

87

public function edit(Job \$job): view

90

}

91

92

// @desc Update a Job Listing

93

// @route PUT /jobs/{id}/

94

public function update(Request \$request, Job \$job): RedirectResponse

95

{

96

// Check if the user is authorized

97

\$this->authorize('update', \$job);

98

99

// Validate the incoming request data

100

\$validatedData = \$request->validate([

101

'title' => 'required|string|max:255',

102

'description' => 'required|string',

103

'salary' => 'required|integer',

104

'tags' => 'nullable|string',

105

'job_type' => 'required|string',

In the update method I can check if the user is authorized

EXPLORER

...

WORKOPIA

app

Http

Controllers

Controller.php

HomeController.php

JobController.php

LoginController.php

RegisterController.php

Middleware

Models

Policies

JobPolicy.php

Providers

AppServiceProvider.php

AuthServiceProvider.php

View

bootstrap

config

database

JobController.php X

JobPolicy.php

AuthServiceProvider.php

Show.blade.php

app > Http > Controllers > JobController.php > JobController > destroy

13

94

class JobController extends Controller

public function update(Request \$request, Job \$job): RedirectResponse

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

}

}

// @desc Delete a Job Listing

// @route DELETE /jobs/{id}

public function destroy(Job \$job): RedirectResponse

{

// Check if the user is authorized

\$this->authorize('delete', \$job);

// If there is a company logo, delete it from storage

if (\$job->company_logo) {

Storage::delete('public/logos/' . \$job->company_logo);

}

// Delete the job

\$job->delete();

return redirect()->route('jobs.index')->with('success', 'Job listing deleted successfully!');

}

In the destroy method I can check if the user is authorized

However, If I am logged in as test user, I can still access the job edit form of a job I don't own.

Edit Job Listing

Job Info

EXPLORER

WORKOPIA

app

Http

Controllers

Controller.php

HomeController.php

JobController.php

LoginController.php

RegisterController.php

Middleware

Models

JobController.php

JobPolicy.php

AuthServiceProvider.php

Show.blade.php

app > Http > Controllers > JobController.php > JobController

```
13 class JobController extends Controller
83
84
85 // @desc Edit a Job form
86 // @route GET /jobs/{id}/edit
87 public function edit(Job $job): view
88 {
89 // Check if the user is authorized
90 $this->authorize('update', $job);
91
92 return view('jobs.edit')->with('job', $job);
93 }
```

In the edit method I can check if the user is authorized

10. Dashboard, Profile & Pagination

[10.1 Intro](#)

[10.2 Dashboard Controller & View](#)

[10.3 Dashboard user Job Listings](#)

[10.4 Profile Controller & Info Update](#)

[10.5 Profile Avatar upload](#)

[10.6 Show Avatar in Header](#)

[10.7 Simple Job Pagination](#)

[10.8 Customise Pagination View](#)

10.1 Intro



Laravel From Scratch

Dashboard, Profile & Pagination - Section Intro

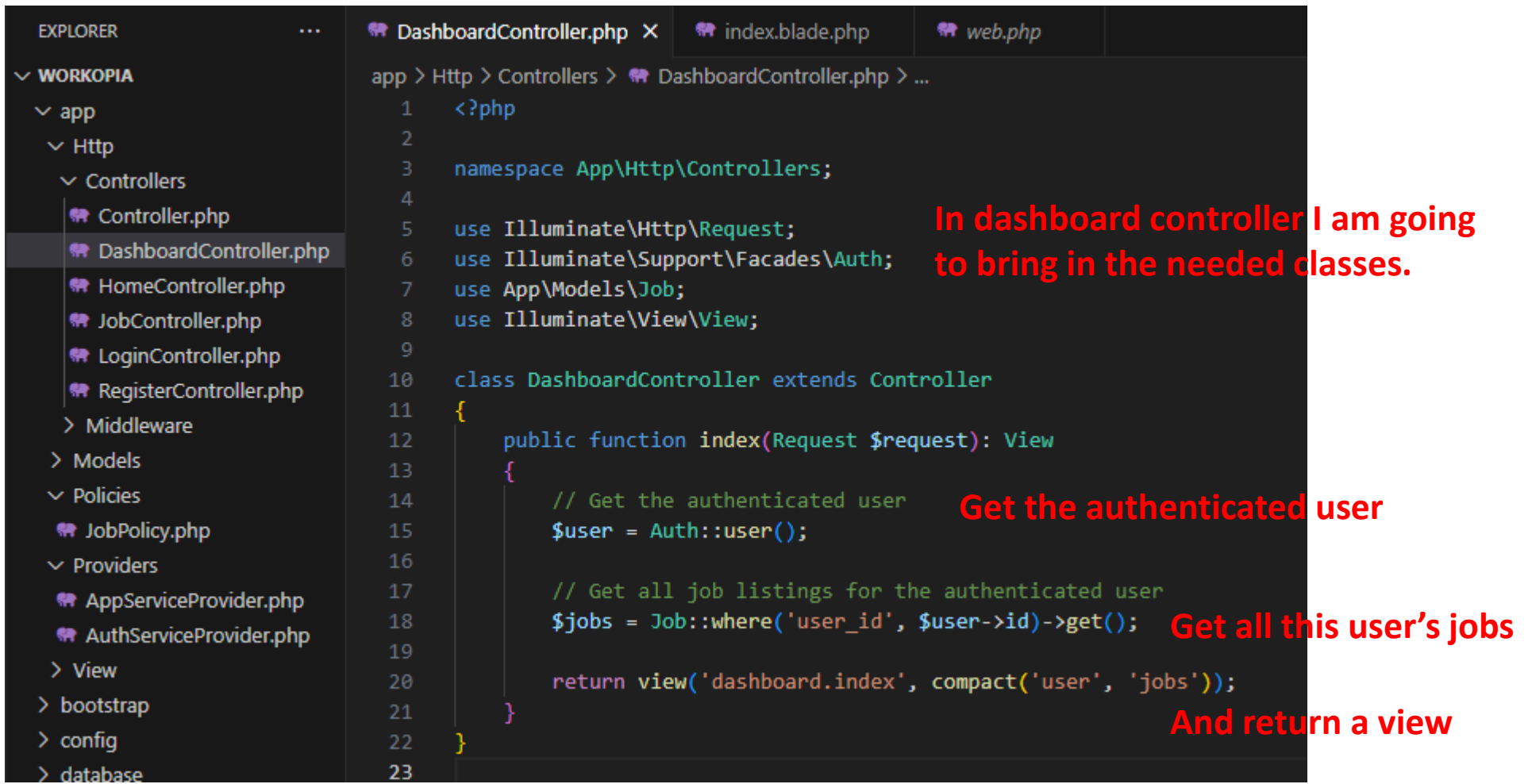
- ✓ **Dashboard Controller/View**
- ✓ **User Job Listings**
- ✓ **Profile Controller/Update**
- ✓ **User Avatar Upload**
- ✓ **Job Pagination**
- ✓ **Customize Pagination Links**

10.2 Dashboard Controller & View

```
PS C:\Users\ellio\Herd\workopia> php artisan make:controller DashboardController
```

```
INFO Controller [C:\Users\ellio\Herd\workopia\app\Http\Controllers\DashboardController.php] created successfully.
```

```
PS C:\Users\ellio\Herd\workopia>
```



The screenshot shows an IDE with a file explorer on the left and a code editor on the right. The file explorer shows the project structure with 'WORKOPIA' as the root, containing 'app', 'bootstrap', 'config', and 'database' folders. The 'app' folder is expanded, showing 'Http' and 'Controllers' subfolders. The 'DashboardController.php' file is selected in the 'Controllers' folder. The code editor shows the content of 'DashboardController.php' with the following code:

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use Illuminate\Support\Facades\Auth;
7 use App\Models\Job;
8 use Illuminate\View\View;
9
10 class DashboardController extends Controller
11 {
12     public function index(Request $request): View
13     {
14         // Get the authenticated user
15         $user = Auth::user();
16
17         // Get all job listings for the authenticated user
18         $jobs = Job::where('user_id', $user->id)->get();
19
20         return view('dashboard.index', compact('user', 'jobs'));
21     }
22 }
23
```

Annotations in red text are placed next to specific lines of code:

- In dashboard controller I am going to bring in the needed classes.** (Next to lines 5-8)
- Get the authenticated user** (Next to line 15)
- Get all this user's jobs** (Next to line 18)
- And return a view** (Next to line 20)

EXPLORER

- WORKOPIA
 - resources
 - views
 - pages
 - index.blade.php
 - layout.blade.php
 - welcome.blade.php
 - routes
 - console.php
 - web.php
 - storage
 - tests
 - vendor
 - .editorconfig
 - .env
 - .env.example
 - .gitattributes
 - .gitignore
 - .md
 - artisan

JobController.php DashboardController.php web.php X JobPolicy.php AuthServiceProvider.php Show.blade.php

routes > web.php > ...

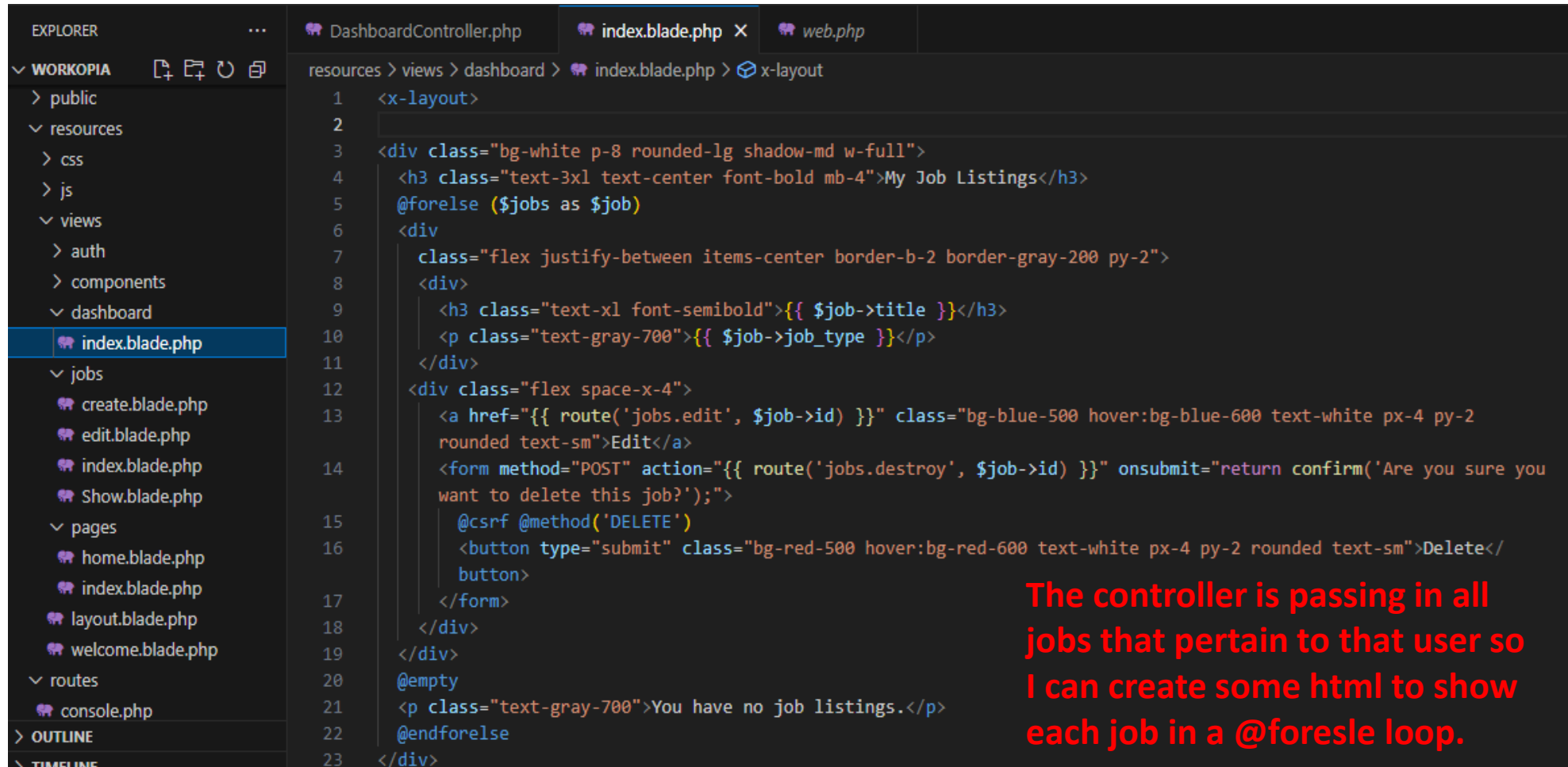
```
5 use App\Http\Controllers\HomeController;
6 use App\Http\Controllers\LoginController;
7 use App\Http\Controllers\RegisterController;
8 use App\Http\Controllers\DashboardController;
9
10 Route::get('/', [HomeController::class, 'index'])->name('home');
11 // Route::resource('jobs', JobController::class);
12 Route::resource('jobs', JobController::class)->middleware('auth')->only(['create', 'edit', 'update', 'destroy']);
13 Route::resource('jobs', JobController::class)->except(['create', 'edit', 'update', 'destroy']);
14
15 Route::middleware('guest')->group(function () {
16     Route::get('/register', [RegisterController::class, 'register'])->name('register');
17     Route::post('/register', [RegisterController::class, 'store'])->name('register.store');
18     Route::get('/login', [LoginController::class, 'login'])->name('login');
19     Route::post('/login', [LoginController::class, 'authenticate'])->name('login.authenticate');
20 });
21
22 Route::post('/logout', [LoginController::class, 'logout'])->name('logout');
23
24 Route::get('/dashboard', [DashboardController::class, 'index'])->name('dashboard')->middleware('auth');
25
```

Bring in the dashboard controller to the routing file

Add a route to the dashboard using the Auth middleware

Create in resources\views\ a folder called 'dashboard' and a file within called index.blade.php

10.3 Dashboard User Job Listings



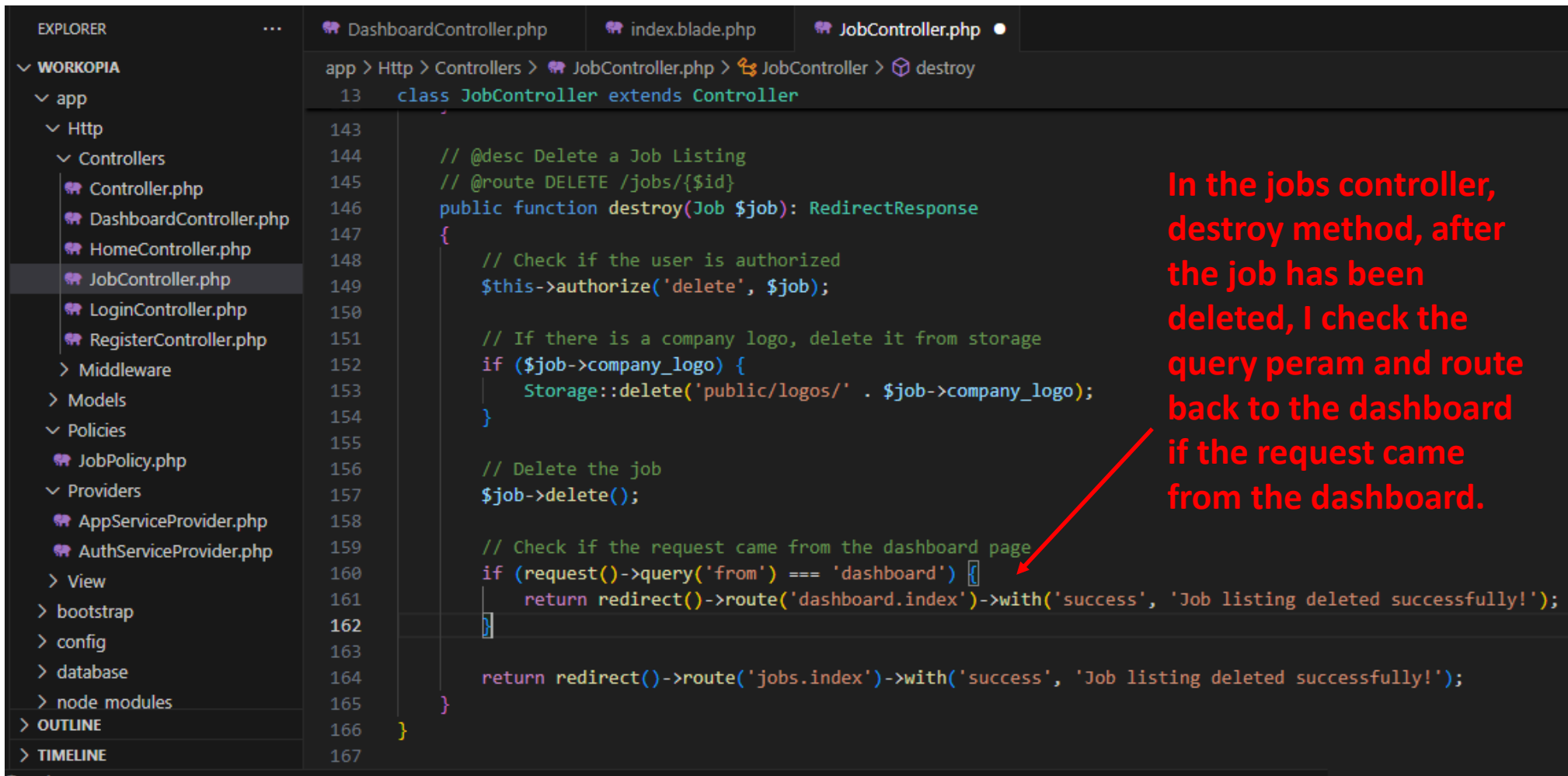
The screenshot shows a code editor with the Explorer panel on the left and the editor area on the right. The Explorer panel shows the project structure with folders like public, resources, css, js, views, auth, components, dashboard, jobs, pages, routes, and OUTLINE. The dashboard folder is expanded, showing index.blade.php selected. The editor area shows the content of index.blade.php, which is a Blade template for displaying job listings. The template includes a header, a main content area with a flex container for job listings, and a footer. The job listings are displayed in a table with columns for title, type, and actions (edit and delete). The delete action is a form with a CSRF token and a DELETE method.

```
1 <x-layout>
2
3 <div class="bg-white p-8 rounded-lg shadow-md w-full">
4   <h3 class="text-3xl text-center font-bold mb-4">My Job Listings</h3>
5   @forelse ($jobs as $job)
6     <div
7       class="flex justify-between items-center border-b-2 border-gray-200 py-2">
8       <div>
9         <h3 class="text-xl font-semibold">{{ $job->title }}</h3>
10        <p class="text-gray-700">{{ $job->job_type }}</p>
11      </div>
12      <div class="flex space-x-4">
13        <a href="{{ route('jobs.edit', $job->id) }}" class="bg-blue-500 hover:bg-blue-600 text-white px-4 py-2
14          rounded text-sm">Edit</a>
15        <form method="POST" action="{{ route('jobs.destroy', $job->id) }}" onsubmit="return confirm('Are you sure you
16          want to delete this job?');">
17          @csrf @method('DELETE')
18          <button type="submit" class="bg-red-500 hover:bg-red-600 text-white px-4 py-2 rounded text-sm">Delete</
19          button>
20        </form>
21      </div>
22    </div>
23  @empty
24    <p class="text-gray-700">You have no job listings.</p>
25  @endforelse
26 </div>
```

The controller is passing in all jobs that pertain to that user so I can create some html to show each job in a @forelse loop.

```
<form method="POST" action="{ route('jobs.destroy', $job->id) }"?from=dashboard"
onsubmit="return confirm('Are you sure you want to delete this job?');">
```

I have modified the form for the delete button to add a query sting to end of the post request - `?from=dashboard` to the end of the route so that when the form is submitted it contains this additional query string.



The screenshot shows a code editor with the following structure:

- EXPLORER
 - WORKOPIA
 - app
 - Http
 - Controllers
 - Controller.php
 - DashboardController.php
 - HomeController.php
 - JobController.php (selected)
 - LoginController.php
 - RegisterController.php
 - Middleware
 - Models
 - Policies
 - JobPolicy.php
 - Providers
 - AppServiceProvider.php
 - AuthServiceProvider.php
 - View
 - bootstrap
 - config
 - database
 - node modules
 - OUTLINE
 - TIMELINE

The main editor displays the `JobController.php` file, showing the `destroy` method:

```
13 class JobController extends Controller
143
144 // @desc Delete a Job Listing
145 // @route DELETE /jobs/{id}
146 public function destroy(Job $job): RedirectResponse
147 {
148     // Check if the user is authorized
149     $this->authorize('delete', $job);
150
151     // If there is a company logo, delete it from storage
152     if ($job->company_logo) {
153         Storage::delete('public/logos/' . $job->company_logo);
154     }
155
156     // Delete the job
157     $job->delete();
158
159     // Check if the request came from the dashboard page
160     if (request()->query('from') === 'dashboard') {
161         return redirect()->route('dashboard.index')->with('success', 'Job listing deleted successfully!');
162     }
163
164     return redirect()->route('jobs.index')->with('success', 'Job listing deleted successfully!');
165 }
166
167 }
```

In the jobs controller, destroy method, after the job has been deleted, I check the query peram and route back to the dashboard if the request came from the dashboard.

10.4 Profile Controller & Info Update

I need to add a form on the dashboard page that shows the user's info and I want to be able to update that info. I am going to create a separate controller for this because it relates more to the user "profile".

```
PS C:\Users\ellio\Herd\workopia> php artisan make:controller ProfileController
```

```
INFO  Controller [C:\Users\ellio\Herd\workopia\app\Http\Controllers\ProfileController.php] created successfully.
```

```
PS C:\Users\ellio\Herd\workopia>
```

EXPLORER

WORKOPIA

app

Http

Controllers

Controller.php

DashboardController.php

HomeController.php

JobController.php

LoginController.php

ProfileController.php

RegisterController.php

Middleware

Models

Policies

JobPolicy.php

Providers

AppServiceProvider.php

AuthServiceProvider.php

View

bootstrap

config

database

OUTLINE

DashboardController.php

index.blade.php

ProfileController.php X

web.php

JobControl

app > Http > Controllers > ProfileController.php > ProfileController > update

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use App\Models\User;
7 use Illuminate\Support\Facades\Auth;
8 use Illuminate\Http\RedirectResponse;
9
10 class ProfileController extends Controller
11 {
12
13     // @desc Update profile info
14     // @route PUT /profile
15     public function update(Request $request): RedirectResponse
16     {
17         // Get the authenticated user
18         $user = Auth::user();
19
20         // Validate the incoming request data
21         $validatedData = $request->validate([
22             'name' => 'required|string|max:255',
23             'email' => 'required|string|email|max:255|unique:users,email,' . $user->id,
24         ]);
25
26         // Update the user profile info
27         $user->name = $request->name;
28         $user->email = $request->email;
29         $user->save();
30
31         return redirect('/profile');
32     }
33 }
```

In profileController, I bring in all the needed classes

I create an update method and define the user as the authenticated user.

EXPLORER

WORKOPIA

app

Http

Controllers

Controller.php

DashboardController.php

HomeController.php

JobController.php

LoginController.php

ProfileController.php

RegisterController.php

Middleware

Models

Policies

JobPolicy.php

Providers

AppServiceProvider.php

AuthServiceProvider.php

View

bootstrap

config

database

OUTLINE

DashboardController.php

index.blade.php

ProfileController.php X

web.php

JobController.php

app > Http > Controllers > ProfileController.php > ProfileController > update

10

class ProfileController extends Controller

11

{

12

13

// @desc Update profile info

14

// @route PUT /profile

15

public function update(Request \$request): RedirectResponse

16

{

17

// Get the authenticated user

18

\$user = Auth::user();

19

20

// Validate the incoming request data

21

\$validatedData = \$request->validate([

22

'name' => 'required|string|max:255',

23

'email' => 'required|string|email|max:255|unique:users,email,' . \$user->id,

24

]);

25

26

// Update the user's information

27

/** @disregard P1013 undefined method */

28

\$user->update(\$validatedData);

29

30

// Redirect back to the dashboard page with a success message

31

return redirect()->route('dashboard.index')->with('success', 'User info updated successfully!');

32

}

33

}

34

I validate the data from the request

Then update the user in the database

Finally, redirect to the dashboard index with success message

The screenshot shows a code editor with the following components:

- EXPLORER (Left Panel):** Displays the project structure. The `routes` folder is expanded, showing `console.php` and `web.php`. `web.php` is selected.
- Routes (Right Panel):** Shows the content of `web.php` with line numbers 9 to 28. The code defines several routes:
 - Line 9: `use App\Http\Controllers\ProfileController;`
 - Line 11: `Route::get('/', [HomeController::class, 'index'])->name('home');`
 - Line 12: `// Route::resource('jobs', JobController::class);`
 - Line 13: `Route::resource('jobs', JobController::class)->middleware('auth')->only(['create', 'edit', 'update', 'destroy']);`
 - Line 14: `Route::resource('jobs', JobController::class)->except(['create', 'edit', 'update', 'destroy']);`
 - Line 16: `Route::middleware('guest')->group(function () {`
 - Line 17: `Route::get('/register', [RegisterController::class, 'register'])->name('register');`
 - Line 18: `Route::post('/register', [RegisterController::class, 'store'])->name('register.store');`
 - Line 19: `Route::get('/login', [LoginController::class, 'login'])->name('login');`
 - Line 20: `Route::post('/login', [LoginController::class, 'authenticate'])->name('login.authenticate');`
 - Line 21: `});`
 - Line 23: `Route::post('/logout', [LoginController::class, 'logout'])->name('logout');`
 - Line 25: `Route::get('/dashboard', [DashboardController::class, 'index'])->name('dashboard.index')->middleware('auth');`
 - Line 27: `Route::put('/profile', [ProfileController::class, 'update'])->name('profile.update')->middleware('auth');`
 - Line 28: An empty line.

A red arrow points to the `'update'` parameter in the `Route::put` method on line 27. Below the code, a red text overlay states:

I need a route to the update method that is a PUT and with authentication middleware

I also fixed the dashboard route so that it goes to dashboard.index

User info updated successfully!

Profile Info

Name

Test Use

Email

test@test.com

When I update the user, it also changes in the database

My Job Listings

Software Engineer

Part-Time

Edit

Delete

Marketing Specialist

Full-Time

Edit

Delete

| | id
[PK] bigint | name
character varying (255) | email
character varying (255) | email_verified_at
timestamp without time zone |
|---|-------------------|---------------------------------|----------------------------------|--|
| 1 | 1 | Test Use | test@test.com | 2025-06-25 16:35:15 |
| 2 | 2 | Zoie Hintz | beatty.adolfo@example.com | 2025-06-25 16:35:15 |

Job listing deleted successfully!

Profile Info

Name

Test Use

Email

test@test.com

If I delete a job from the dashboard, it correctly redirects back to the dashboard and displays my job listings with the job removed.

My Job Listings

Marketing Specialist

Full-Time

Edit

Delete

Finally, I reseed the database and verify that 'test user' has two jobs

```
welcome.blade.php
26
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\ellio\Herd\workopia> php artisan db:seed

INFO Seeding database.

Database\Seeders\TestUserSeeder ..... RUNNING
Database\Seeders\TestUserSeeder ..... 319 ms DONE

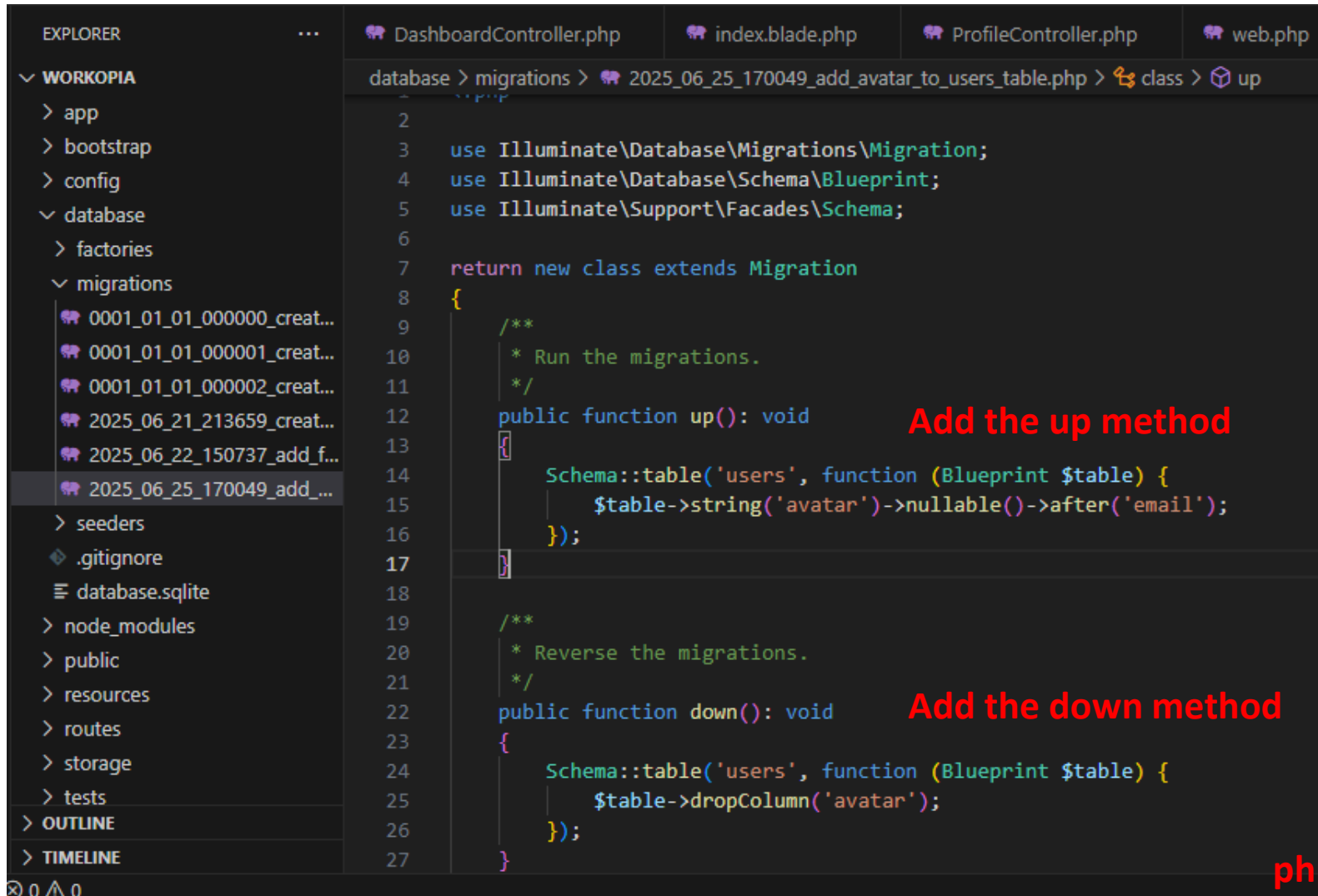
Database\Seeders\RandomUserSeeder ..... RUNNING
Users Created Successfully Database\Seeders\RandomUserSeeder ..... 311 ms DONE

Database\Seeders\JobSeeder ..... RUNNING
Database\Seeders\JobSeeder ..... 10 ms DONE

PS C:\Users\ellio\Herd\workopia>
```

10.5 Profile Avatar Upload

I decided that I want to have users have the ability to upload an avatar. Right now, from the dashboard, we can see the user's name and email. I want to add an avatar to this as well. Let's add the avatar upload to the form. **php artisan make:migration add_avatar_to_users_table --table=users**



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like 'app', 'bootstrap', 'config', 'database', 'factories', 'migrations', 'seeders', and files like '.gitignore', 'database.sqlite', 'node_modules', 'public', 'resources', 'routes', 'storage', 'tests'. The 'migrations' folder is expanded, showing several migration files. The file '2025_06_25_170049_add_avatar_to_users_table.php' is selected. The code editor shows the content of this migration file. The file starts with a docblock comment, followed by 'use' statements for 'Illuminate\Database\Migrations\Migration', 'Illuminate\Database\Schema\Blueprint', and 'Illuminate\Support\Facades\Schema'. Then, a 'return new class extends Migration' statement is followed by an opening curly brace. Inside the class, there is a docblock comment for the 'up' method, followed by a 'public function up(): void' statement. The 'up' method contains a 'Schema::table' call for the 'users' table, with a function that adds a nullable 'avatar' column after the 'email' column. This is followed by a closing curly brace for the 'up' method. Then, there is a docblock comment for the 'down' method, followed by a 'public function down(): void' statement. The 'down' method contains a 'Schema::table' call for the 'users' table, with a function that drops the 'avatar' column. This is followed by a closing curly brace for the 'down' method. The file ends with a closing curly brace for the class. Red annotations are present: 'Add the up method' points to the 'up' method definition, and 'Add the down method' points to the 'down' method definition. At the bottom right, the text 'php artisan migrate' is written in red.

```
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 return new class extends Migration
8 {
9     /**
10      * Run the migrations.
11      */
12     public function up(): void
13     {
14         Schema::table('users', function (Blueprint $table) {
15             $table->string('avatar')->nullable()->after('email');
16         });
17     }
18
19     /**
20      * Reverse the migrations.
21      */
22     public function down(): void
23     {
24         Schema::table('users', function (Blueprint $table) {
25             $table->dropColumn('avatar');
26         });
27     }
28 }
```

php artisan migrate

Columns



| | | Name | Data type | Length/Precision | Scale | Not NULL? | Primary key? | Default |
|--|--|------------------|-----------------------|------------------|-------|-------------------------------------|-------------------------------------|------------|
| | | id | bigint v | | | ☒ | ☒ | nextval('i |
| | | name | character varying v | 255 | | ☒ | ☐ | |
| | | email | character varying v | 255 | | ☒ | ☐ | |
| | | email_verified_a | timestamp withou v | 0 | | ☐ | ☐ | |
| | | password | character varying v | 255 | | ☒ | ☐ | |
| | | remember_token | character varying v | 100 | | ☐ | ☐ | |
| | | created_at | timestamp withou v | 0 | | ☐ | ☐ | |
| | | updated_at | timestamp withou v | 0 | | ☐ | ☐ | |
| | | avatar | character varying v | 255 | | ☐ | ☐ | |



✕ Close

↺ Reset

💾 Save

Avatar column has been added to the user table

The screenshot shows an IDE with a file explorer on the left and a code editor on the right. The file explorer is expanded to show the 'Models' directory, with 'User.php' selected. The code editor displays the 'User' class, which extends 'Authenticatable'. The '\$fillable' array is being updated to include 'avatar'. A red arrow points from a text annotation to the 'avatar' field in the array.

```
EXPLORER    ...    Controller.php    index.blade.php    ProfileController

WORKOPIA
├── app
│   ├── Http
│   └── Models
│       ├── Job.php
│       └── User.php
├── Policies
├── Providers
├── View
├── bootstrap
├── config
├── database
│   └── factories
└── ...

app > Models > User.php > User > $fillable
11  class User extends Authenticatable
18
19      * @var list<string>
20      */
21  protected $fillable = [
22      'name',
23      'email',
24      'password',
25      'avatar',
26  ];
27
28  /**
29   * The attributes that should be hidden from users.
30  */
```

Update the user model to include the avatar field

EXPLORER

WORKOPIA

resources

views

components

button-link.blade.php

header.blade.php

hero.blade.php

job-card.blade.php

logout-button.blade.php

nav-link.blade.php

topbanner.blade.php

dashboard

index.blade.php

jobs

create.blade.php

edit.blade.php

index.blade.php

Show.blade.php

pages

home.blade.php

index.blade.php

OUTLINE

TIMELINE

index.blade.php X

edit.blade.php

ProfileController.php

RegisterController.php

web.php

User.php

Job.php

resources > views > dashboard > index.blade.php > x-layout > section.flex.flex-col.md:flex-row.gap-6 > div.bg-white.p-8.rounded-lg.shadow-md.w-f

1 <x-layout>

3 <section class="flex flex-col md:flex-row gap-6">

4

5 <div class="bg-white p-8 rounded-lg shadow-md w-full md:w-1/2">

6 <h3 class="text-3xl text-center font-bold mb-4">Profile Info</h3>

7

8 @if(\$user->avatar)

9 <div class="mt-2 flex justify-center">

10

12 </div>

13 @endif

14 <form method="POST" action="{{ route('profile.update') }}" enctype="multipart/form-data">

15 @csrf

16 @method('PUT')

17

18 <!-- profile Name -->

19 <x-inputs.text id="name" name="name" label="Name" placeholder="Name" :value="old('name', \$user->name)"/>

20

21 <!-- Profile Email -->

22 <x-inputs.text id="email" name="email" label="Email" type="email" placeholder="Email" :value="old('email',

23 \$user->email)"/>

24

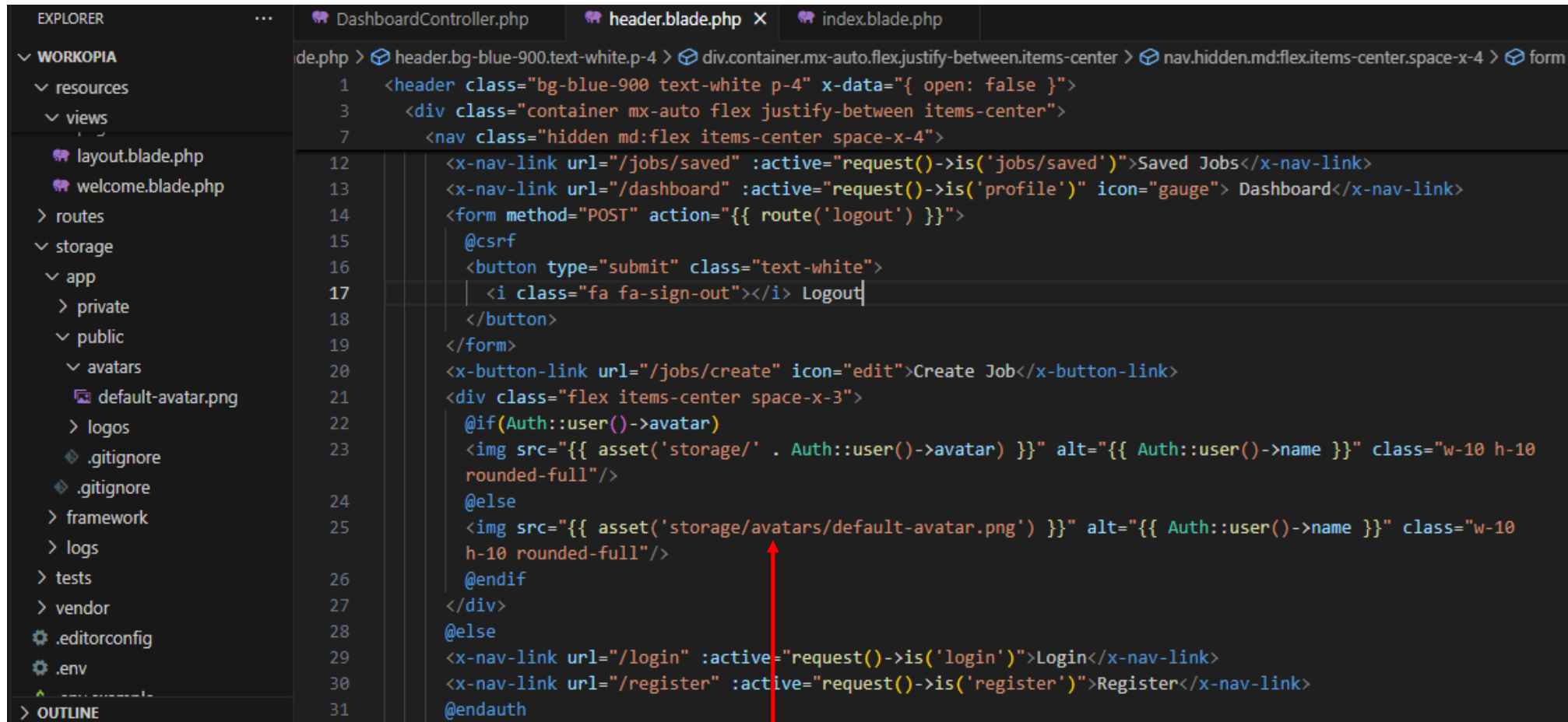
25 <!-- Company Logo -->

<x-inputs.file id="avatar" name="avatar" label="Profile Avatar"/>

If the user has an avatar image, display it

Clean up the user form reusing the input components

10.6 Show Avatar in Header



```
1 <header class="bg-blue-900 text-white p-4" x-data="{ open: false }">
3 <div class="container mx-auto flex justify-between items-center">
7 <nav class="hidden md:flex items-center space-x-4">
12 <x-nav-link url="/jobs/saved" :active="request()->is('jobs/saved')">Saved Jobs</x-nav-link>
13 <x-nav-link url="/dashboard" :active="request()->is('profile')" icon="gauge"> Dashboard</x-nav-link>
14 <form method="POST" action="{{ route('logout') }}">
15     @csrf
16     <button type="submit" class="text-white">
17         <i class="fa fa-sign-out"></i> Logout
18     </button>
19 </form>
20 <x-button-link url="/jobs/create" icon="edit">Create Job</x-button-link>
21 <div class="flex items-center space-x-3">
22     @if(Auth::user()->avatar)
23         name }}" class="w-10 h-10 rounded-full"/>
24     @else
25         name }}" class="w-10 h-10 rounded-full"/>
26     @endif
27 </div>
28 @else
29 <x-nav-link url="/login" :active="request()->is('login')">Login</x-nav-link>
30 <x-nav-link url="/register" :active="request()->is('register')">Register</x-nav-link>
31 @endauth
```

In the header blade I have an @if directive that checks if the authenticated user has an avatar. If there is an avatar it displays the image, if no avatar then it displays a default image.

Workopia | find and list jobs x default avatar - Buscar con Goo x +

workopia.test/dashboard

Workopia

Home All Jobs Saved Jobs Dashboard Logout Create Job

Profile Info

Name

Test User

My Job Listings

Software Engineer
Part-Time

Edit Delete

I reseeded the database then deleted the avatar in the storage\avatar folder and uploaded the default image. Now for test user, it displays the default image. When I upload an avatar, it shows in the dashboard and the header

Workopia

Home All Jobs Saved Jobs Dashboard Logout Create Job

Profile Info



My Job Listings

Software Engineer
Part-Time

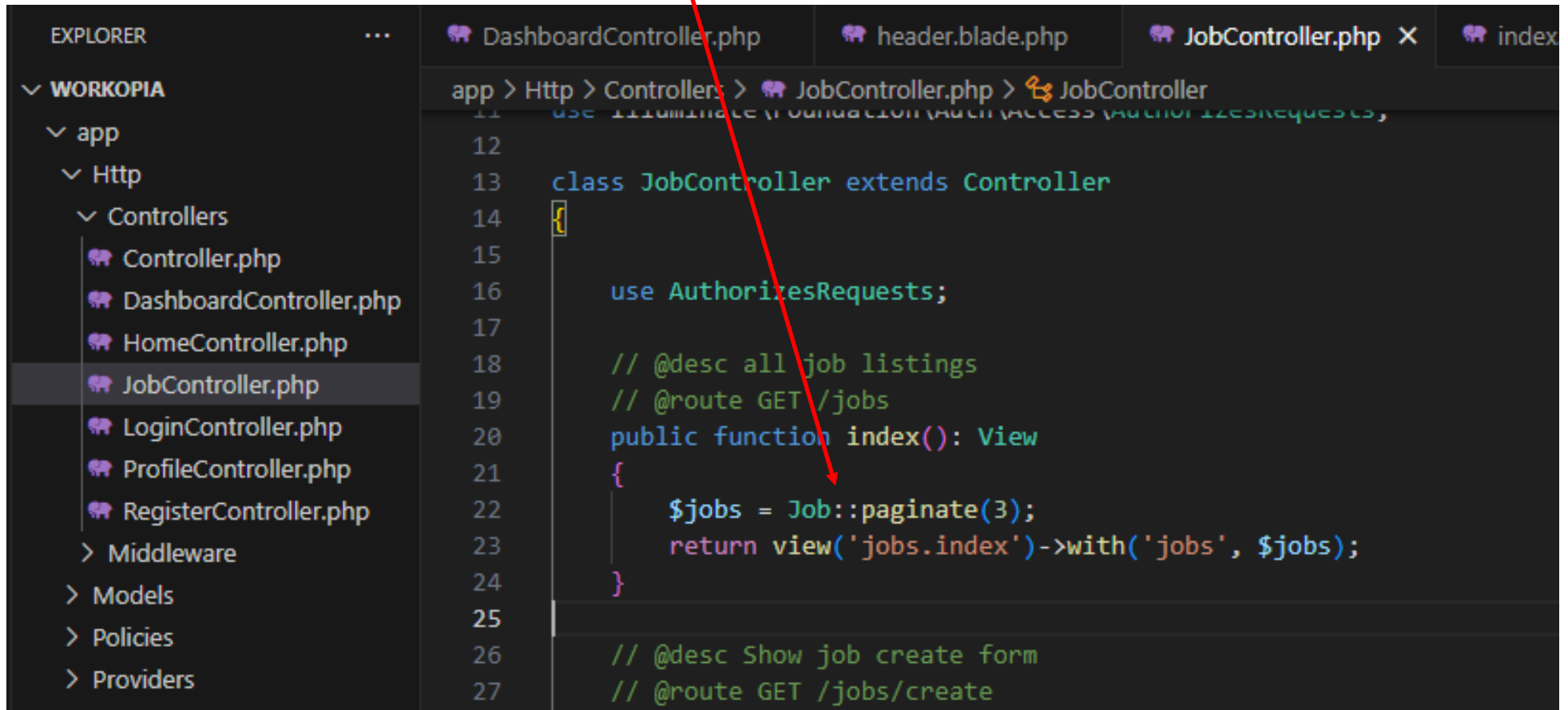
Marketing Specialist
Full-Time

Edit Delete

Edit Delete

10.7 Simple Job Pagination

Laravel makes simple job pagination... simple!. If I change `$jobs = Jobs::All()` to `$jobs = Job::paginate(3)` then it only shows 3 jobs on the 'all jobs' page. I can view the jobs through the query params i.e. `jobs/?page=2`



```
EXPLORER
WORKOPIA
  app
    Http
      Controllers
        Controller.php
        DashboardController.php
        HomeController.php
        JobController.php
        LoginController.php
        ProfileController.php
        RegisterController.php
      Middleware
      Models
      Policies
      Providers

DashboardController.php
header.blade.php
JobController.php X
index

app > Http > Controllers > JobController.php > JobController
11 use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
12
13 class JobController extends Controller
14 {
15
16     use AuthorizesRequests;
17
18     // @desc all job listings
19     // @route GET /jobs
20     public function index(): View
21     {
22         $jobs = Job::paginate(3);
23         return view('jobs.index')->with('jobs', $jobs);
24     }
25
26     // @desc Show job create form
27     // @route GET /jobs/create
```

```
resources > views > jobs > index.blade.php > ...
1  <x-layout>
2  <x-slot name="title">All Jobs</x-slot>
3  <h1 class="text-center text-3xl mb-4 font-bold border border-gray-300 p-3">All Jobs</h1>
4
5  <div class="grid grid-cols-1 md:grid-cols-3 gap-4 mb-6">
6      @forelse($jobs as $job)
7          <x-job-card :job="$job" />
8      @empty
9          <p>No jobs found</p>
10     @endforelse
11 </div>
12
13 <!-- Pagination Links -->
14 <div class="mt-4">{{ $jobs->links() }}</div>
15
16 </x-layout>
17
18
```

Now I can add Laravel pagination links at the bottom of the all jobs page

All Jobs



Data Analyst

Full-Time

Quantum Code is in search of a Data Analyst to join our team and transform complex data into actiona...

Salary: 75,000

Location: Chicago, IL On-site

Tags: Data Analysis, Statistics

Details



Graphic Designer

Full-Time

As a Graphic Designer at Shield, you will be at the forefront of our creative projects, working on d...

Salary: 70,000

Location: Albany, NY On-site

Tags: Graphic Design, Creative

Details



Data Scientist

Full-Time

Join Sparkle as a Data Scientist and play a pivotal role in analyzing and interpreting complex datas...

Salary: 100,000

Location: Boston, MA Remote

Tags: Data Science, Machine Learning

Details

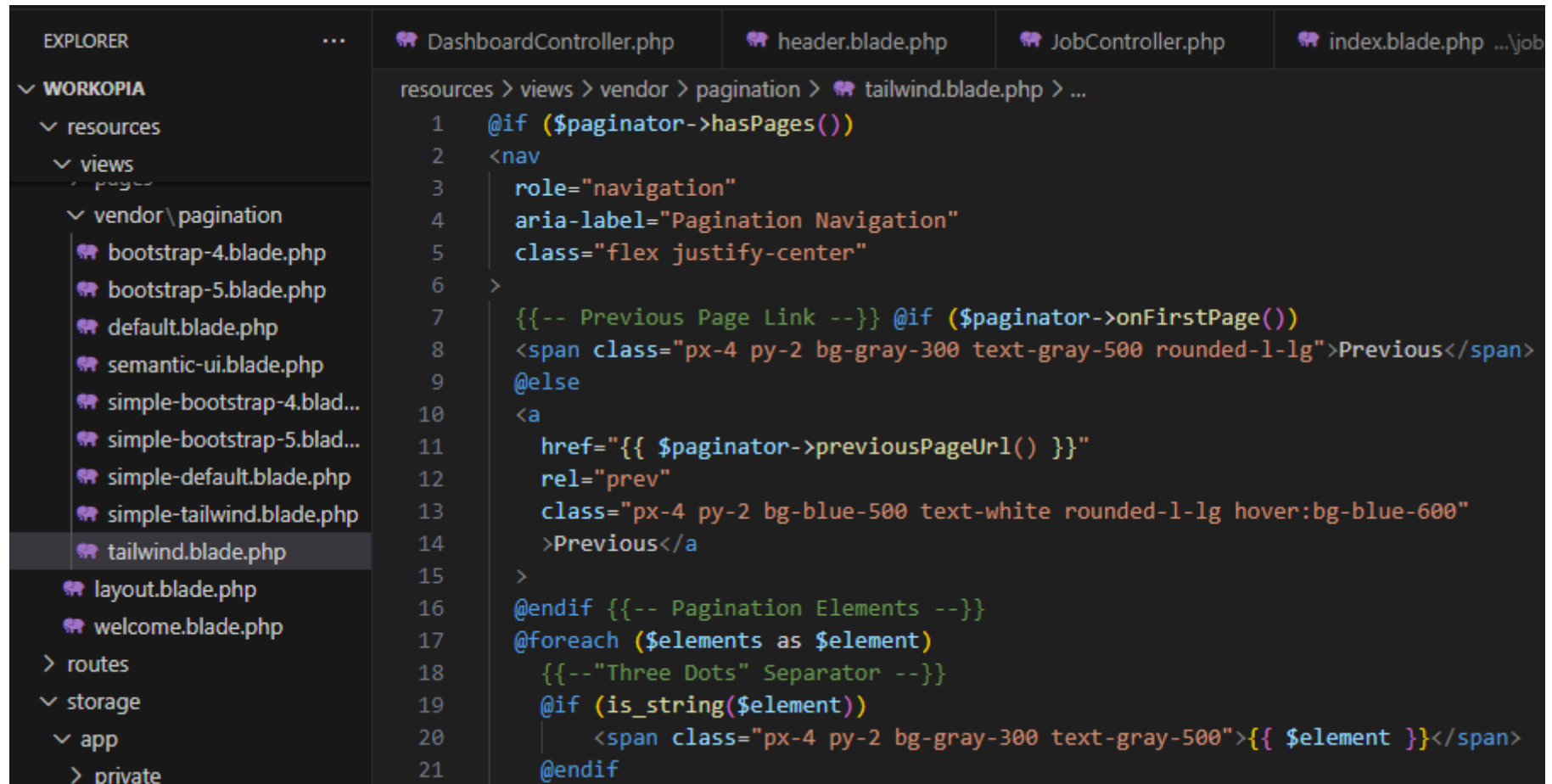
Showing 4 to 6 of 10 results

Laravel takes care of the rest!

10.8 Customise Pagination View

```
artisan vendor:publish --tag=laravel-pagination
```

This will publish the pagination view into the `resources/views/vendor/pagination` directory. There will be multiple files in there and it depends on what CSS framework you are using. In this case Tailwind CSS, so I will edit the `tailwind.blade.php` file. In here I remove the default code and place custom pagination code.



The screenshot shows an IDE with the Explorer panel on the left and the Code editor on the right. The Explorer panel shows the project structure with the following folders and files:

- WORKOPIA
 - resources
 - views
 - vendor\pagination
 - bootstrap-4.blade.php
 - bootstrap-5.blade.php
 - default.blade.php
 - semantic-ui.blade.php
 - simple-bootstrap-4.blad...
 - simple-bootstrap-5.blad...
 - simple-default.blade.php
 - simple-tailwind.blade.php
 - tailwind.blade.php
- layout.blade.php
- welcome.blade.php
- routes
- storage
 - app
 - private

The Code editor shows the content of the `tailwind.blade.php` file, which is a custom pagination view. The code is as follows:

```
resources > views > vendor > pagination > tailwind.blade.php > ...
1  @if ($paginator->hasPages())
2  <nav
3      role="navigation"
4      aria-label="Pagination Navigation"
5      class="flex justify-center"
6  >
7      {{-- Previous Page Link --}} @if ($paginator->onFirstPage())
8      <span class="px-4 py-2 bg-gray-300 text-gray-500 rounded-l-lg">Previous</span>
9      @else
10     <a
11         href="{{ $paginator->previousPageUrl() }}"
12         rel="prev"
13         class="px-4 py-2 bg-blue-500 text-white rounded-l-lg hover:bg-blue-600"
14     >Previous</a>
15     >
16 @endif {{-- Pagination Elements --}}
17 @foreach ($elements as $element)
18     {{-- "Three Dots" Separator --}}
19     @if (is_string($element))
20         <span class="px-4 py-2 bg-gray-300 text-gray-500">{{ $element }}</span>
21     @endif
```




IT Support Specialist

Full-Time

Tec Solutions is hiring an IT Support Specialist to provide technical assistance and support to our...

Salary: 65,000

Location: Dallas, TX On-site

Tags: IT Support, Networking

Details

Now there is custom pagination.



Previous

1

2

3

4

Next

11. Bookmark Job Listings

[11.1 Intro](#)

[11.2 Bookmark Migration & Relationships](#)

[11.3 Seeding Bookmarks](#)

[11.4 Get & Show Bookmarks](#)

[11.5 Bookmarking Jobs](#)

[11.6 Removing Bookmarks](#)

11.1 Intro



Laravel From Scratch

Bookmark Job Listings - Section Intro

- ✓ **Bookmark Table Migration**
- ✓ **Seeding Bookmarks**
- ✓ **Fetch & Display Bookmarks**
- ✓ **Bookmark Button**
- ✓ **Remove Bookmark Button**

11.2 Bookmarks Migration & Relationships

I need to create a foreign key table that links the jobs table with the user's table. It is a table that is used to store the relationship between two other tables.

`php artisan make:migration create_job_user_bookmarks_table`

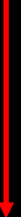
```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('job_user_bookmarks', function (Blueprint $table) {
            $table->id();
            $table->foreignId('user_id')->constrained()->onDelete('cascade');
            $table->foreignId('job_id')->constrained('job_listings')->onDelete('cascade');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('job_user_bookmarks');
    }
};
```

The table contains an id row, a row with the id of the user, a row with the id of the job and the two standard timestamp rows.



php artisan migrate

job_user_bookmarks ×

General Columns Advanced Constraints Partitions Parameters Security SQL

Inherited from table(s) ▼

Columns +

| | | Name | Data type | Length/Precision | Scale | Not NULL? | Primary key? | Default |
|--|--|---|---|------------------|-------|-------------------------------------|-------------------------------------|---|
| | | <input type="text" value="id"/> | <input type="text" value="bigint"/> ▼ | | | ☒ | ☒ | <input type="text" value="nextval('j"/> |
| | | <input type="text" value="user_id"/> | <input type="text" value="bigint"/> ▼ | | | ☒ | ☐ | <input type="text"/> |
| | | <input type="text" value="job_id"/> | <input type="text" value="bigint"/> ▼ | | | ☒ | ☐ | <input type="text"/> |
| | | <input type="text" value="created_at"/> | <input type="text" value="timestamp without"/> ▼ | 0 | | ☐ | ☐ | <input type="text"/> |
| | | <input type="text" value="updated_at"/> | <input type="text" value="timestamp without"/> ▼ | 0 | | ☐ | ☐ | <input type="text"/> |

Running the `php artisan migrate` command will create the table. Now it needs to be seeded.

EXPLORER

- WORKOPIA
 - app
 - Http
 - Controllers
 - Controller.php
 - DashboardController.php
 - HomeController.php
 - JobController.php
 - LoginController.php
 - ProfileController.php
 - RegisterController.php
 - Middleware
 - Models
 - Job.php
 - User.php
 - Policies

header.blade.php JobController.php Job.php User.php index.blade.php ...jobs tai

app > Models > Job.php > Job

```
16 use Illuminate\Database\Eloquent\Relations\BelongsTo;
37
38
39 // Relation to user
40 public function user(): BelongsTo
41 {
42     return $this->belongsTo(User::class);
43 }
44
45 // Relation to Bookmarks
46 public function bookmarkedByUsers(): BelongsToMany
47 {
48     return $this->belongsToMany(User::class, 'job_user_bookmarks')->withTimestamps();
49 }
50 }
51
```

In the model for 'job' I bring in the BelongsTo class and define the relationship between the 'job_listings' table and the 'job_user_bookmark'

EXPLORER

- WORKOPIA
 - app
 - Http
 - Controllers
 - Middleware
 - Models
 - Job.php
 - User.php
 - Policies
 - Providers
 - View
 - bootstrap
 - config
 - database

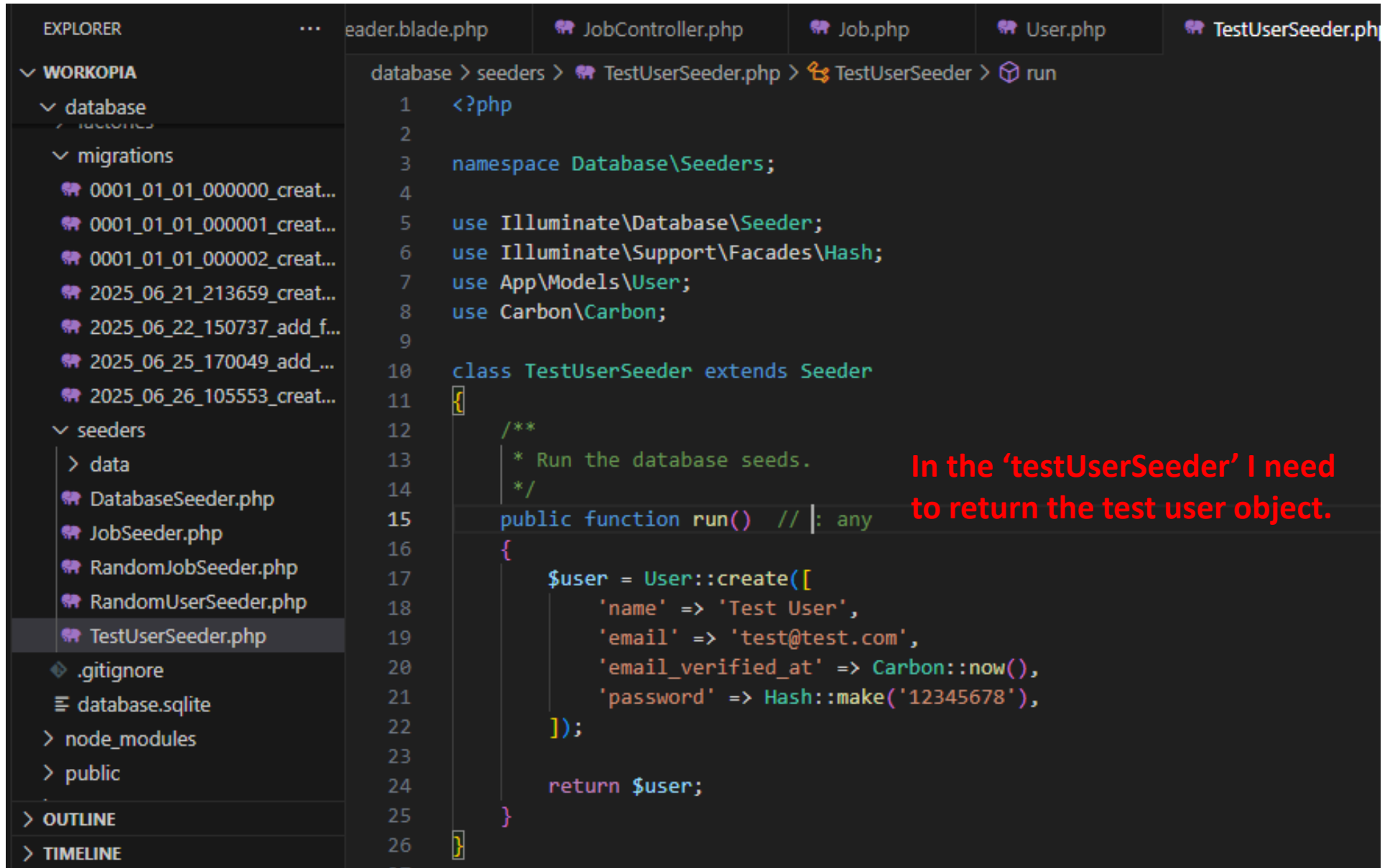
header.blade.php JobController.php Job.php User.php index.blade.php ...jobs tailw

app > Models > User.php > ...

```
12 class User extends Authenticatable
51
52 // Relate job listing to user
53 public function jobListings(): HasMany
54 {
55     return $this->hasMany(Job::class);
56 }
57
58 // Relate User to Job Listing
59 public function bookmarkedJobs(): BelongsToMany
60 {
61     return $this->belongsToMany(Job::class, 'job_user_bookmarks')->withTimestamps();
62 }
63 }
64
```

In the model for user' I bring in the BelongsToMany class and define the relationship between the 'user' table and the 'job_user_bookmark'

11.3 Seeding Bookmarks



The screenshot shows the Visual Studio Code interface with the Explorer panel on the left and the Editor panel on the right. The Explorer panel shows the project structure, including the `database/seeders` directory. The Editor panel displays the `TestUserSeeder.php` file, which contains the following code:

```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6 use Illuminate\Support\Facades\Hash;
7 use App\Models\User;
8 use Carbon\Carbon;
9
10 class TestUserSeeder extends Seeder
11 {
12     /**
13      * Run the database seeds.
14      */
15     public function run() // |: any
16     {
17         $user = User::create([
18             'name' => 'Test User',
19             'email' => 'test@test.com',
20             'email_verified_at' => Carbon::now(),
21             'password' => Hash::make('12345678'),
22         ]);
23
24         return $user;
25     }
26 }
```

In the 'testUserSeeder' I need to return the test user object.

```
<?php
```

```
php artisan make:seeder BookmarkSeeder
```

```
namespace Database\Seeders;
```

```
use Illuminate\Database\Console\Seeds\WithoutModelEvents;
```

```
use Illuminate\Database\Seeder;
```

```
use App\Models\User;
```

```
use App\Models\Job;
```

```
class BookmarkSeeder extends Seeder
```

```
{
```

```
    /**
```

```
     * Run the database seeds.
```

```
     */
```

```
    public function run(): void
```

```
    {
```

```
        // Get the test user
```

```
        $testUser = User::where('email', 'test@test.com')->firstOrFail();
```

```
        // Get all job IDs
```

```
        $jobIds = Job::pluck('id')->toArray();
```

```
        // Randomly select job IDs to bookmark
```

```
        $randomJobIds = array_rand($jobIds, 3); // Will seed 3 jobs to test user
```

```
        // Attach the selected jobs as bookmarks for the test user
```

```
        foreach ($randomJobIds as $jobId) {
```

```
            $testUser->bookmarkedJobs()->attach($jobIds[$jobId]);
```

```
        }
```

```
    }
```

```
}
```

In the bookmark seeder, I get the test user, then randomly select three jobs, then insert three rows into the bookmarks foreign key table with the attach method


```
<?php
```

```
namespace Database\Seeders;
```

```
// use Illuminate\Database\Console\Seeds\WithoutModelEvents;
```

```
use Illuminate\Database\Seeder;
```

```
use Illuminate\Support\Facades\DB;
```

```
class DatabaseSeeder extends Seeder
```

```
{
```

```
    /**
```

```
     * Seed the application's database.
```

```
     */
```

```
    public function run(): void
```

```
    {
```

```
        // Truncate tables
```

```
        DB::table('job_listings')->truncate();
```

```
        DB::table('users')->truncate();
```

```
        DB::table('job_user_bookmarks')->truncate();
```

```
        $this->call(TestUserSeeder::class);
```

```
        $this->call(RandomUserSeeder::class);
```

```
        $this->call(JobSeeder::class);
```

```
        $this->call(BookmarkSeeder::class);
```

```
    }
```

```
}
```

I modify the database seeder to truncate the bookmarks foreign key table and to also run the bookmarking seeding function.

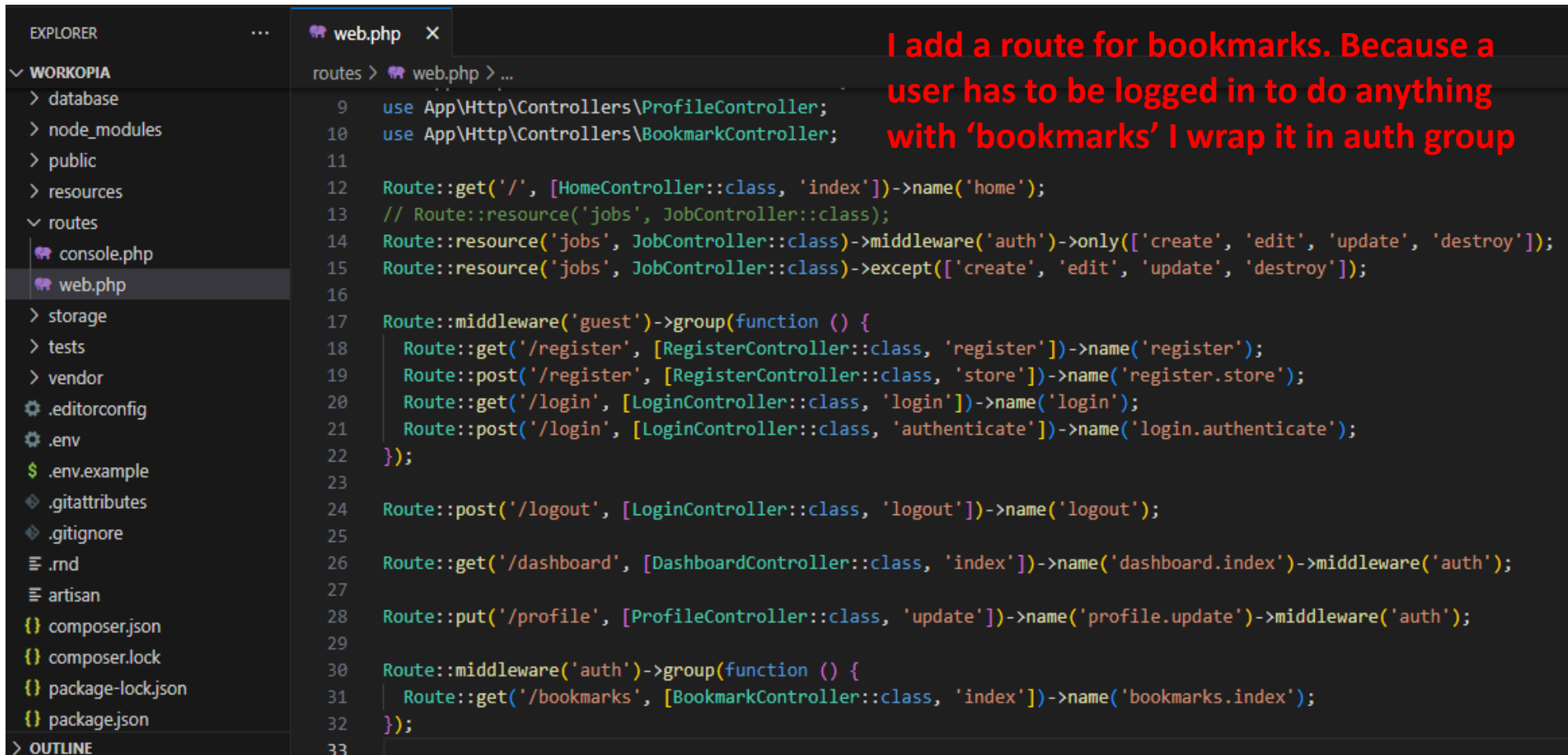
```
php artisan db:seed
```

| | id
[PK] bigint  | user_id
bigint  | job_id
bigint  | created_at
timestamp without time zone  | updated_at
timestamp without time zone  |
|---|---|---|--|---|---|
| 1 | 1 | 1 | 2 | 2025-06-26 11:56:13 | 2025-06-26 11:56:13 |
| 2 | 2 | 1 | 4 | 2025-06-26 11:56:13 | 2025-06-26 11:56:13 |
| 3 | 3 | 1 | 10 | 2025-06-26 11:56:13 | 2025-06-26 11:56:13 |

The job_user_bookmark table has been seeded with three rows

11.4 Get & Show Bookmarks

```
php artisan make:controller BookmarkController
```



```
9 use App\Http\Controllers\ProfileController;
10 use App\Http\Controllers\BookmarkController;
11
12 Route::get('/', [HomeController::class, 'index'])->name('home');
13 // Route::resource('jobs', JobController::class);
14 Route::resource('jobs', JobController::class)->middleware('auth')->only(['create', 'edit', 'update', 'destroy']);
15 Route::resource('jobs', JobController::class)->except(['create', 'edit', 'update', 'destroy']);
16
17 Route::middleware('guest')->group(function () {
18     Route::get('/register', [RegisterController::class, 'register'])->name('register');
19     Route::post('/register', [RegisterController::class, 'store'])->name('register.store');
20     Route::get('/login', [LoginController::class, 'login'])->name('login');
21     Route::post('/login', [LoginController::class, 'authenticate'])->name('login.authenticate');
22 });
23
24 Route::post('/logout', [LoginController::class, 'logout'])->name('logout');
25
26 Route::get('/dashboard', [DashboardController::class, 'index'])->name('dashboard.index')->middleware('auth');
27
28 Route::put('/profile', [ProfileController::class, 'update'])->name('profile.update')->middleware('auth');
29
30 Route::middleware('auth')->group(function () {
31     Route::get('/bookmarks', [BookmarkController::class, 'index'])->name('bookmarks.index');
32 });
33
```

I add a route for bookmarks. Because a user has to be logged in to do anything with 'bookmarks' I wrap it in auth group

File Edit Selection View Go Run ... workopia

EXPLORER

WORKOPIA

- app
 - Http
 - Controllers
 - BookmarkController.php**
 - Controller.php
 - DashboardController.php
 - HomeController.php
 - JobController.php
 - LoginController.php
 - ProfileController.php
 - RegisterController.php
 - Middleware
 - Models
 - Job.php
 - User.php
 - Policies
 - Providers
 - View
 - bootstrap
 - config
 - database
- OUTLINE

web.php

BookmarkController.php X

JobController.php

ProfileContro

app > Http > Controllers > BookmarkController.php > BookmarkController > index

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\RedirectResponse;
6 use Illuminate\View\View;
7 use Illuminate\Support\Facades\Auth;
8
9 use Illuminate\Http\Request;
10
11 class BookmarkController extends Controller
12 {
13     // @desc    Get all users Bookmarks
14     // @route    GET /bookmarks
15     public function index() // : View
16     {
17         $user = Auth::user();
18
19         /** @disregard p1013 Undefined method */
20         $bookmarks = $user->bookmarkedJobs()->paginate(10);
21
22         dd($bookmarks);
23     }
24 }
25
```

In the bookmark controller I get the user from the authenticated user

I call the user model to get the bookmarked jobs

EXPLORER...WORKOPIApublicresourcescssjsviewsauthcomponentsinputsalert.blade.phpbottom-banner.blade.phpbutton-link.blade.phpheader.blade.phphero.blade.phpjob-card.blade.php

resources > views > components > header.blade.php > header.bg-blue-900.text-white.p-4 > div.container.mx-auto.flex.justify-between.items-center

```
1 <header class="bg-blue-900 text-white p-4" x-data="{ open: false }">
2
3 <div class="container mx-auto flex justify-between items-center">
4   <h1 class="text-3xl font-semibold">
5     <x-nav-link url="/" :active="request()->is('/')">Workopia</x-nav-link>
6   </h1>
7   <nav class="hidden md:flex items-center space-x-4">
8
9     <x-nav-link url="/" :active="request()->is('/')">Home</x-nav-link>
10    <x-nav-link url="/jobs" :active="request()->is('jobs')">All Jobs</x-nav-link>
11    @auth
12      <x-nav-link url="/bookmarks" :active="request()->is('bookmarks')">Saved Jobs</x-nav-link>
13    <x-nav-link url="/dashboard" :active="request()->is('profile') icon="gauge"> Dashboard</x-nav-link>
14    <form method="POST" action="{{ route('logout') }}">
15      @csrf
16      <button type="submit" class="text-white">
```

In the header blade I ensure that the saved jobs link is pointing to the bookmarks view

Illuminate\Pagination\LengthAwarePaginator {#1277 ▼ // app\Http\Controllers\BookmarkController.php:21

#items: Illuminate\Database\Eloquent\Collection {#1272 ▼

#items: array:3 [▼

0 => App\Models\Job {#1269 ►}

1 => App\Models\Job {#1278 ►}

2 => App\Models\Job {#1279 ►}

]

#escapeWhenCastingToString: false }

#perPage: 10

#currentPage: 1

#path: <https://workopia.test/bookmarks>

#query: [] #fragment: null #pageName: "page"

#escapeWhenCastingToString: false +onEachSide: 3

#options: array:2 [►]

#total: 3

#lastPage: 1 }

When I am logged in and click on the saved jobs link in the header it lands on the die dump where I see that I am returning an array of three bookmarked jobs.

EXPLORER

WORKOPIA

app

Http

Controllers

BookmarkController.php

Controller.php

DashboardController.php

HomeController.php

JobController.php

LoginController.php

ProfileController.php

RegisterController.php

Middleware

Models

Job.php

User.php

Policies

Providers

View

bootstrap

config

database

web.php

BookmarkController.php

JobController.php

ProfileController.php

app > Http > Controllers > BookmarkController.php > BookmarkController > index

```
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\RedirectResponse;
6 use Illuminate\View\View;
7 use Illuminate\Support\Facades\Auth;
8
9 use Illuminate\Http\Request;
10
11 class BookmarkController extends Controller
12 {
13     // @desc    Get all users Bookmarks
14     // @route    GET /bookmarks
15     public function index() // : View
16     {
17         $user = Auth::user();
18
19         /** @disregard p1013 Undefined method */
20         $bookmarks = $user->bookmarkedJobs()->paginate(9);
21
22         return view('jobs.bookmark')->With('bookmarks', $bookmarks);
23     }
24 }
25
```

Now in the bookmarkController,
I want to return a view to
job.bookmarks passing in the
bookmarks data

EXPLORER

WORKOPIA

resources

views

components

header.blade.php

hero.blade.php

job-card.blade.php

logout-button.blade.php

nav-link.blade.php

topbanner.blade.php

dashboard

jobs

bookmarked.blade.php

create.blade.php

resources > views > jobs > bookmarked.blade.php > x-layout

```
1 <x-layout>
2   <h2 class="text-center text-3xl mb-4 font-bold border border-gray-300 p-3">Bookmarked Jobs</h2>
3
4   <div class="grid grid-cols-1 md:grid-cols-3 gap-4 mb-6">
5     @forelse ($bookmarks as $bookmark)
6       <x-job-card :job="$bookmark" />
7     @empty
8       <p class="text-center text-gray-500">No bookmarks available</p>
9     @endforelse
10  </div>
11
12  <{{ $bookmarks->links() }}>
13 </x-layout>
```

I create the blade for the bookmarks looping over the array of bookmarks. For each job I want to show a job card so I reuse the blade card component.

workopia.test/bookmarks

Workopia

Home All Jobs Saved Jobs Dashboard Logout Create Job

The view is made. Next, I need to add the un-bookmark and bookmark functionality to the job details page



Marketing Specialist

Full-Time

Bitwave is seeking a creative and strategic Marketing Specialist to develop and execute comprehensiv...



Data Analyst

Full-Time

Quantum Code is in search of a Data Analyst to join our team and transform complex data into actiona...

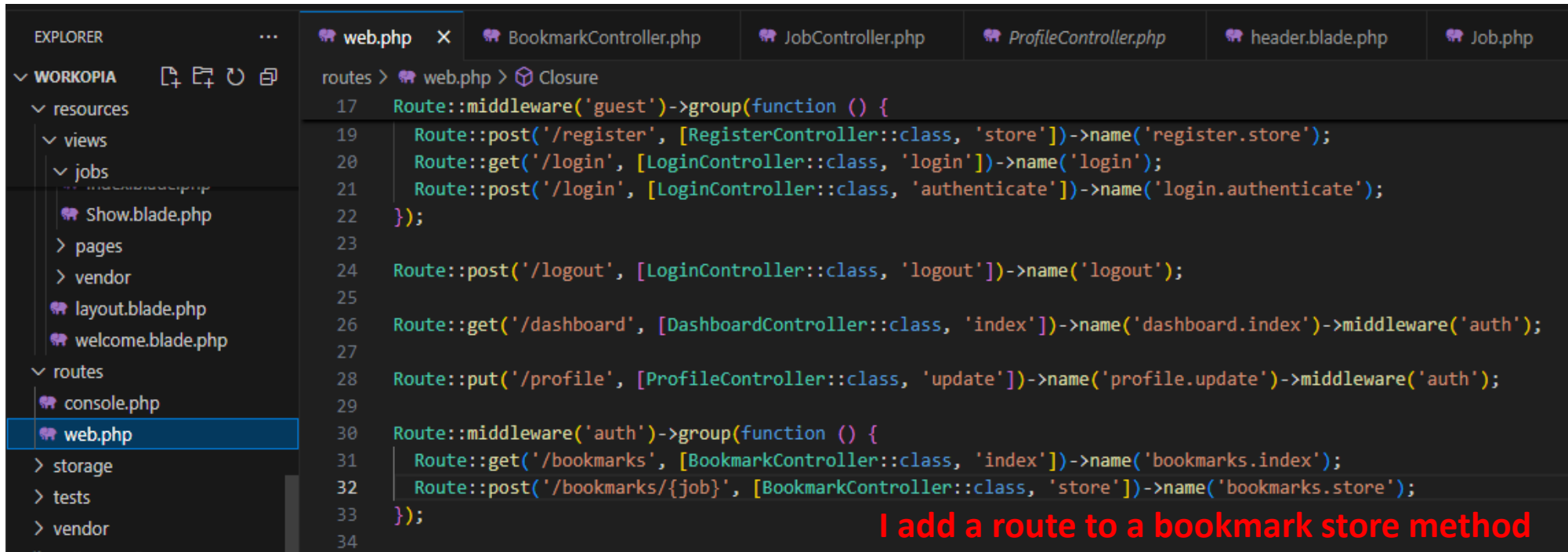


IT Support Specialist

Full-Time

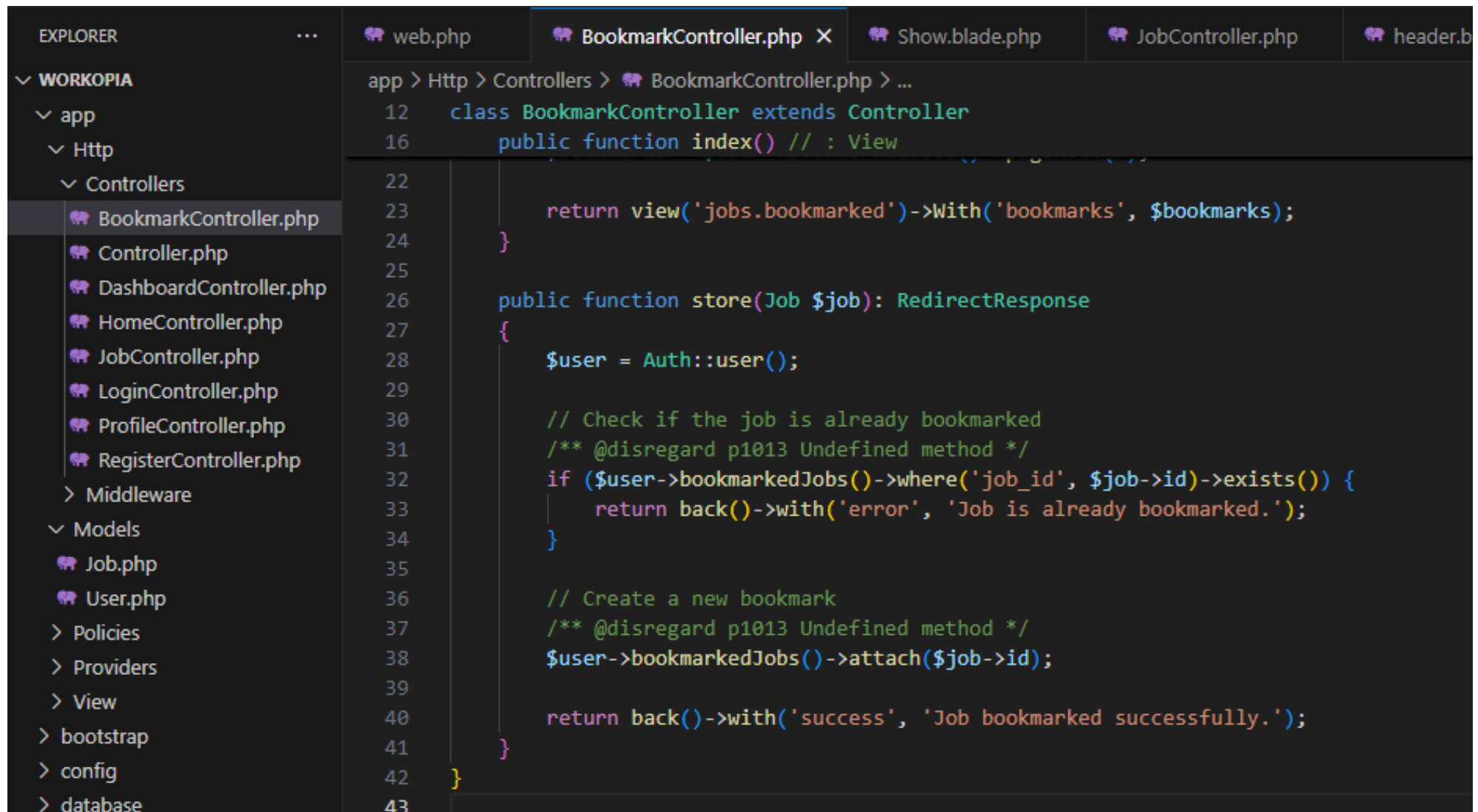
Tec Solutions is hiring an IT Support Specialist to provide technical assistance and support to our...

11.5 Bookmarking Jobs



```
routes > web.php > Closure
17 Route::middleware('guest')->group(function () {
19     Route::post('/register', [RegisterController::class, 'store'])->name('register.store');
20     Route::get('/login', [LoginController::class, 'login'])->name('login');
21     Route::post('/login', [LoginController::class, 'authenticate'])->name('login.authenticate');
22 });
23
24 Route::post('/logout', [LoginController::class, 'logout'])->name('logout');
25
26 Route::get('/dashboard', [DashboardController::class, 'index'])->name('dashboard.index')->middleware('auth');
27
28 Route::put('/profile', [ProfileController::class, 'update'])->name('profile.update')->middleware('auth');
29
30 Route::middleware('auth')->group(function () {
31     Route::get('/bookmarks', [BookmarkController::class, 'index'])->name('bookmarks.index');
32     Route::post('/bookmarks/{job}', [BookmarkController::class, 'store'])->name('bookmarks.store');
33 });
34
```

I add a route to a bookmark store method



```
EXPLORER  ...  web.php  BookmarkController.php X  Show.blade.php  JobController.php  header.b

WORKOPIA
├── app
│   ├── Http
│   │   └── Controllers
│   │       ├── BookmarkController.php
│   │       ├── Controller.php
│   │       ├── DashboardController.php
│   │       ├── HomeController.php
│   │       ├── JobController.php
│   │       ├── LoginController.php
│   │       ├── ProfileController.php
│   │       └── RegisterController.php
│   ├── Middleware
│   ├── Models
│   │   ├── Job.php
│   │   └── User.php
│   ├── Policies
│   ├── Providers
│   ├── View
│   ├── bootstrap
│   ├── config
│   └── database

app > Http > Controllers > BookmarkController.php > ...
12  class BookmarkController extends Controller
16      public function index() // : View
17      {
18          // ...
19      }
20
21      return view('jobs.bookmarked')->With('bookmarks', $bookmarks);
22  }
23
24  public function store(Job $job): RedirectResponse
25  {
26      $user = Auth::user();
27
28      // Check if the job is already bookmarked
29      /** @disregard p1013 Undefined method */
30      if ($user->bookmarkedJobs()->where('job_id', $job->id)->exists()) {
31          return back()->with('error', 'Job is already bookmarked.');
```

In the Store method of the bookmark controller, the user is obtained from the authenticated user. I then check if a bookmark already exists in the database and return an error message if true. Otherwise, I attach the bookmark and return a success message.

```
EXPLORER
WORKOPIA
resources
views
components
nav-link.blade.php
topbanner.blade.php
dashboard
jobs
bookmarked.blade.php
create.blade.php
edit.blade.php
index.blade.php
Show.blade.php
pages
vendor
layout.blade.php
welcome.blade.php
routes
console.php
web.php

web.php
BookmarkController.php
Show.blade.php X
JobController.php
header.blade.php
Job.php

resources > views > jobs > Show.blade.php > x-layout > div.grid.grid-cols-1.md:grid-cols-4.gap-6 > aside.bg-white.rounded-lg.shadow-md.p-3 >
1 <x-layout>
2 <div class="grid grid-cols-1 md:grid-cols-4 gap-6">
80 <aside class="bg-white rounded-lg shadow-md p-3">
99
100 @endif
101 @if ($job->company_phone)
102 <p class="text-gray-700 text-lg mt-2">Phone: {{ $job->company_phone }}</p>
103 @endif
104 <!-- Bookmark Button -->
105 @guest
106 <p class="mt-10 bg-gray-200 text-gray-700 font-bold w-full py-2 px-4 rounded-full text-center">
107 <i class="fas fa-info-circle mr-3"></i> You must be logged in to bookmark this job.</p>
108 @else
109 <form
110 action="{{ route('bookmarks.store', $job->id) }}"method="POST" class="mt-10">
111 @csrf
112 <button type="submit" class="bg-blue-500 hover:bg-blue-600 text-white font-bold w-full py-2 px-4
rounded-full flex items-center justify-center"><i class="fas fa-bookmark mr-3"></i> Bookmark Listing</
button>
113 </form>
114 @endguest
115
```

In the show blade, I can use the @guest directive to show a login required message on the bookmark buton and the else clause can show a form with an action to the bookmarks.store method passing in the job Id.



Job is already bookmarked.

[Back To Listings](#)

Company Info

Site Location: San Francisco, CA

Tags: Marketing, Advertising

marketing strategies and results-driven solutions.

[Visit Website](#)

You must be logged in to bookmark this job.

Job Details

Job Requirements

Bachelors degree in Marketing or related field, experience in digital marketing

EXPLORER

WORKOPIA

app

Http

Controllers

BookmarkController.php

Controller.php

DashboardController.php

HomeController.php

JobController.php

LoginController.php

ProfileController.php

RegisterController.php

Middleware

Models

Job.php

web.phpBookmarkController.phpXShow.blade.phpJobController.phpheader.blade.phpJob.php

app > Http > Controllers > BookmarkController.php > BookmarkController

9
10 use Illuminate\Http\Request;
11
12 class BookmarkController extends Controller
13 {
14 // @desc Get all users Bookmarks
15 // @route GET /bookmarks
16 public function index() // : View
17 {
18 \$user = Auth::user();
19
20 /** @disregard p1013 Undefined method */
21 \$bookmarks = \$user->bookmarkedJobs()->orderBy('job_user_bookmarks', 'Created_at', 'DESC')->paginate(9);
22
23 return view('jobs.bookmarked')->with('bookmarks', \$bookmarks);
24 }
25

In the bookmarks controller I use the OrderBy laravel method and pass in the table, column and optional DESC parameters.

EXPLORER

WORKOPIA

app

Http

Controllers

BookmarkController.php

Controller.php

DashboardController.php

HomeController.php

JobController.php

LoginController.php

ProfileController.php

RegisterController.php

Middleware

Models

Job.php

web.phpBookmarkController.phpXShow.blade.phpJobController.phpX

app > Http > Controllers > JobController.php > JobController > index

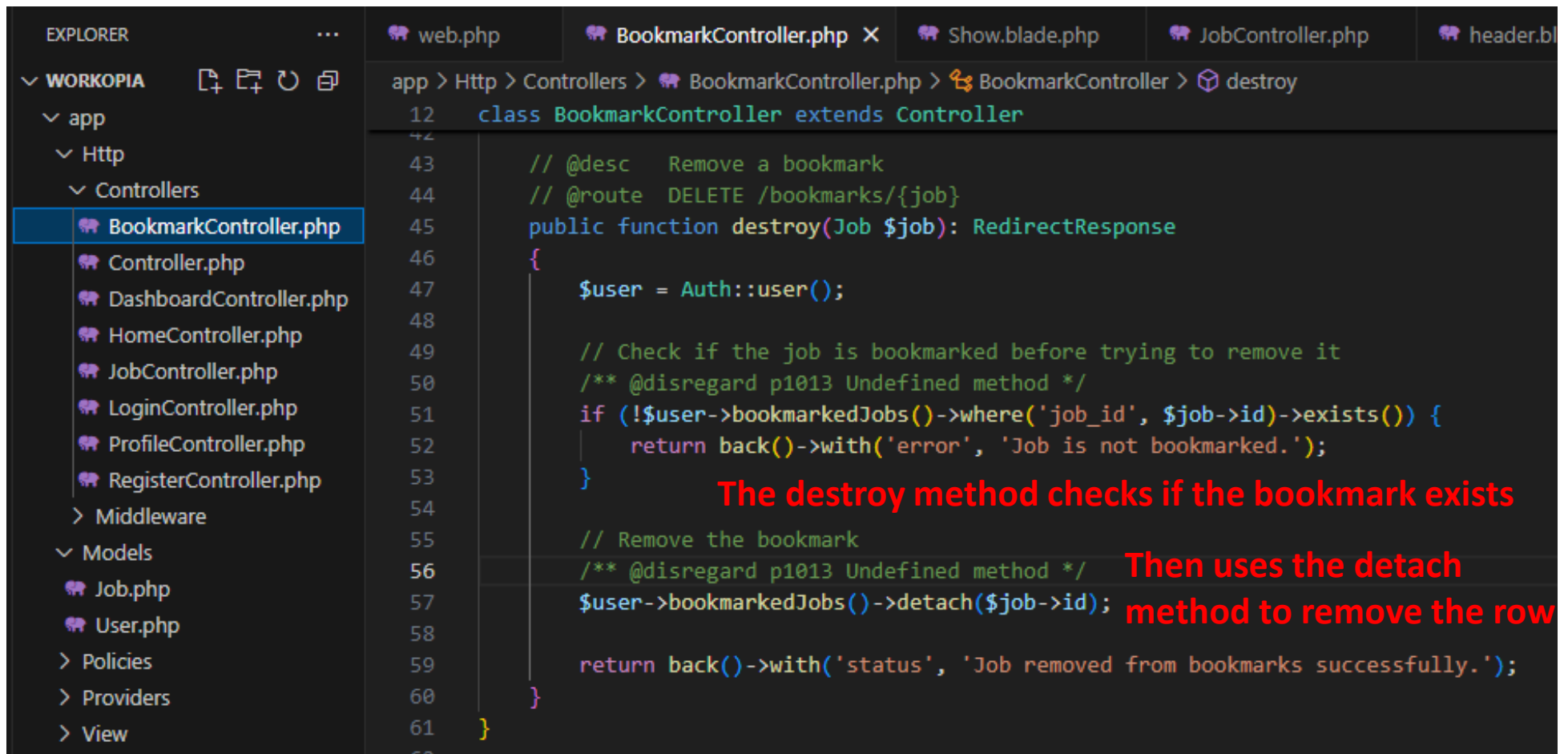
12
13 class JobController extends Controller
14 {
15
16 use AuthorizesRequests;
17
18 // @desc all job listings
19 // @route GET /jobs
20 public function index(): View
21 {
22 \$jobs = Job::latest()->paginate(3);
23 return view('jobs.index')->with('jobs', \$jobs);
24 }
25

In the job controller I use the latest laravel method to order all jobs with the most recent first.

11.6 Remove Bookmarks

I add a route to the destroy method

```
Route::middleware('auth')->group(function () {  
    Route::get('/bookmarks', [BookmarkController::class, 'index'])->name('bookmarks.index');  
    Route::post('/bookmarks/{job}', [BookmarkController::class, 'store'])->name('bookmarks.store');  
    Route::delete('/bookmarks/{job}', [BookmarkController::class, 'destroy'])->name('bookmarks.destroy');  
});
```



The screenshot shows an IDE with the Explorer panel on the left and the Editor panel on the right. The Explorer panel shows the project structure with the following folders and files:

- WORKOPIA
 - app
 - Http
 - Controllers
 - BookmarkController.php** (selected)
 - Controller.php
 - DashboardController.php
 - HomeController.php
 - JobController.php
 - LoginController.php
 - ProfileController.php
 - RegisterController.php
 - Middleware
 - Models
 - Job.php
 - User.php
 - Policies
 - Providers
 - View

The Editor panel shows the code for `BookmarkController.php` with the following content:

```
12 class BookmarkController extends Controller  
43 // @desc Remove a bookmark  
44 // @route DELETE /bookmarks/{job}  
45 public function destroy(Job $job): RedirectResponse  
46 {  
47     $user = Auth::user();  
48  
49     // Check if the job is bookmarked before trying to remove it  
50     /** @disregard p1013 Undefined method */  
51     if (!$user->bookmarkedJobs()->where('job_id', $job->id)->exists()) {  
52         return back()->with('error', 'Job is not bookmarked.');53     }  
54  
55     // Remove the bookmark  
56     /** @disregard p1013 Undefined method */  
57     $user->bookmarkedJobs()->detach($job->id);  
58  
59     return back()->with('status', 'Job removed from bookmarks successfully.');60 }  
61 }  
62
```

Red text annotations are present in the Editor panel:

- The destroy method checks if the bookmark exists** (pointing to lines 51-53)
- Then uses the detach method to remove the row** (pointing to line 57)

WORKOPIA

resources

views

components

regulator.blade.php

nav-link.blade.php

topbanner.blade.php

dashboard

jobs

bookmarked.blade.php

create.blade.php

edit.blade.php

index.blade.php

Show.blade.php

pages

vendor

layout.blade.php

welcome.blade.php

routes

console.php

web.php

storage

OUTLINE

TIMELINE

aside.bg-white.rounded-lg.shadow-md.p-3

form.mt-10

button.bg-blue-500.hover:bg-blue-600.text-white.font-bold.w-full.py-2.px-4.rounded-full.flex.items-center

1 <x-layout>

2 <div class="grid grid-cols-1 md:grid-cols-4 gap-6">

80 <aside class="bg-white rounded-lg shadow-md p-3">

100 @if (\$job->company_phone)

101 <p class="text-gray-700 text-lg mt-2">Phone: {{\$job->company_phone}}</p>

102 @endif

103

104 <!-- Bookmark Button -->

105 @guest

106 <p class="mt-10 bg-gray-200 text-gray-700 font-bold w-full py-2 px-4 rounded-full text-center">

107 | <i class="fas fa-info-circle mr-3"></i> You must be logged in to bookmark this job.</p>

108 @else

109 <form action="{{ {{ auth()->user()->bookmarkedJobs()->where('job_id', \$job->id)->exists() ? route('bookmarks.destroy', \$job->id) : route('bookmarks.store', \$job->id) }}" method="POST" class="mt-10">

110 @csrf

111 @if (auth()->user()->bookmarkedJobs()->where('job_id', \$job->id)->exists())

112 @method('DELETE')

113 <button type="submit" class="bg-red-500 hover:bg-red-600 text-white font-bold w-full py-2 px-4 rounded-full flex items-center justify-center"><i class="fas fa-bookmark mr-3"></i> Remove Bookmark</button>

114 @else

115 <button type="submit" class="bg-blue-500 hover:bg-blue-600 text-white font-bold w-full py-2 px-4 rounded-full flex items-center justify-center"><i class="fas fa-bookmark mr-3"></i> Bookmark Listing</button>

116 @endif

117 </form>

118 @endguest

Now I modify the bookmark button to show you must be logged in or bookmark if job not bookmarked and remove bookmark if job is bookmarked.

Job Type: Full-Time

Remote: No

Salary: \$70,000

Site Location: Albany, NY

Tags: Graphic Design, Creative

Shield is a leading design agency known for its innovative approach to graphic design and creative solutions.

[Visit Website](#)

 **Remove Bookmark**

Job Details

[Job Requirements](#)

12. Job Applicants & Resume Upload

[12.1 Intro](#)

[12.2 Applicant Migration & Model](#)

[12.3 Applicant Form Modal with Alpine.js](#)

[12.4 Fix Modal Blip with x-cloak](#)

[12.5 Applicant Controller & Store Method](#)

[12.6 Show Applications to Owner](#)

[12.7 Delete Applicants](#)

[12.8 Prevent Multiple Applications](#)

12.1 Intro



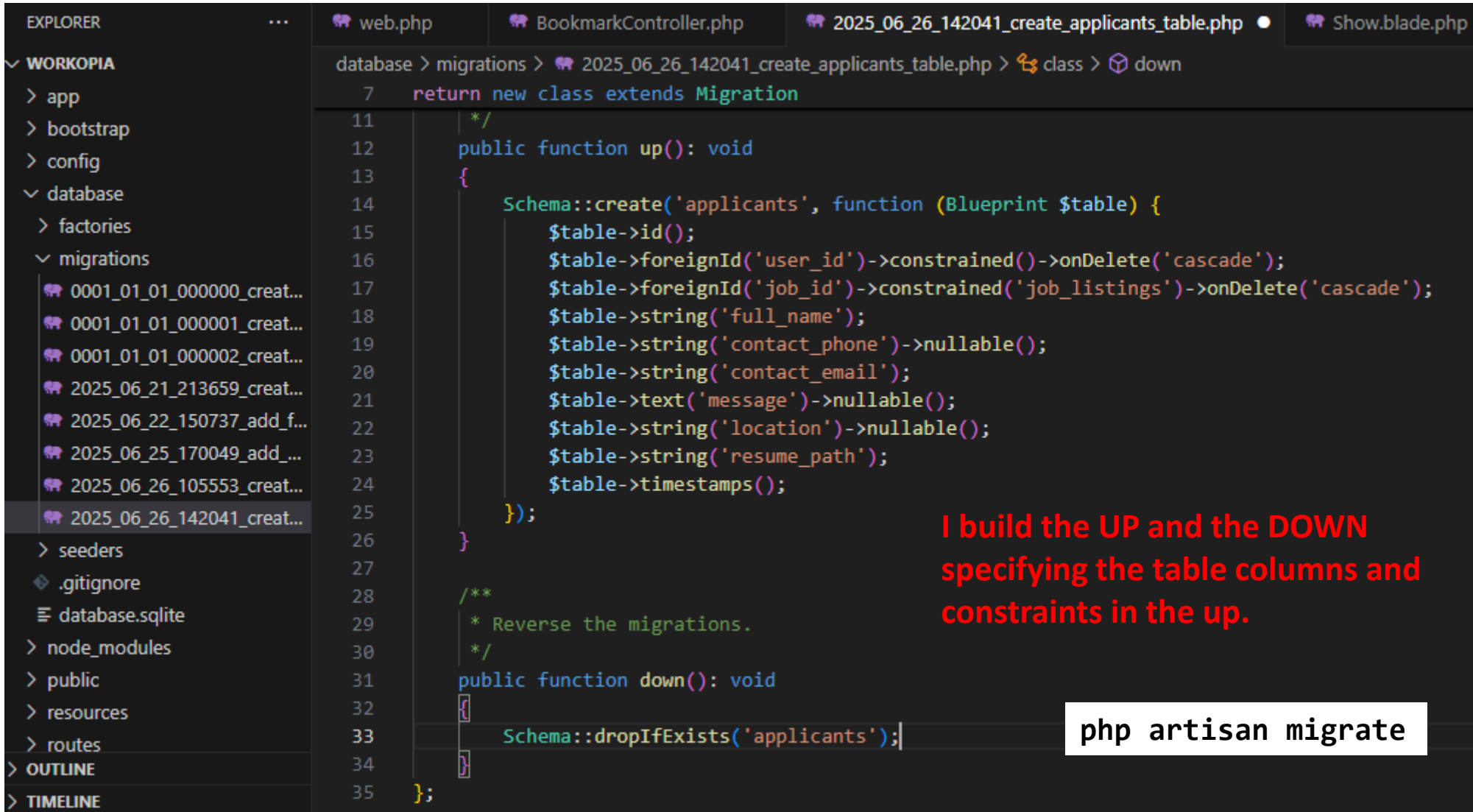
Laravel From Scratch

Job Applicants - Section Intro

- ✓ Applicants Migration & Model
- ✓ Alpine.js Form Modal
- ✓ Applicant Controller
- ✓ Resume Upload
- ✓ Display Applicants
- ✓ Delete Applicants
- ✓ Prevent Multiple Submissions

12.2 Applicant Migration & Model

```
php artisan make:migration create_applicants_table
```



```
EXPLORER  ...  web.php  BookmarkController.php  2025_06_26_142041_create_applicants_table.php  Show.blade.php

WORKOPIA
├── app
├── bootstrap
├── config
├── database
│   ├── factories
│   └── migrations
│       ├── 0001_01_01_000000_creat...
│       ├── 0001_01_01_000001_creat...
│       ├── 0001_01_01_000002_creat...
│       ├── 2025_06_21_213659_creat...
│       ├── 2025_06_22_150737_add_f...
│       ├── 2025_06_25_170049_add_...
│       ├── 2025_06_26_105553_creat...
│       └── 2025_06_26_142041_creat...
├── seeders
├── .gitignore
├── database.sqlite
├── node_modules
├── public
├── resources
├── routes
├── OUTLINE
└── TIMELINE

database > migrations > 2025_06_26_142041_create_applicants_table.php > class > down
7  return new class extends Migration
11  */
12  public function up(): void
13  {
14      Schema::create('applicants', function (Blueprint $table) {
15          $table->id();
16          $table->foreignId('user_id')->constrained()->onDelete('cascade');
17          $table->foreignId('job_id')->constrained('job_listings')->onDelete('cascade');
18          $table->string('full_name');
19          $table->string('contact_phone')->nullable();
20          $table->string('contact_email');
21          $table->text('message')->nullable();
22          $table->string('location')->nullable();
23          $table->string('resume_path');
24          $table->timestamps();
25      });
26  }
27
28  /**
29   * Reverse the migrations.
30   */
31  public function down(): void
32  {
33      Schema::dropIfExists('applicants');
34  }
35  };
```

I build the UP and the DOWN specifying the table columns and constraints in the up.

```
php artisan migrate
```

php artisan make:model Applicant

EXPLORER

WORKOPIA

- app
 - Http
 - Controllers
 - BookmarkController.php
 - Controller.php
 - DashboardController.php
 - HomeController.php
 - JobController.php
 - LoginController.php
 - ProfileController.php
 - RegisterController.php
 - Middleware
 - Models
 - Applicant.php
 - Job.php
 - User.php
 - Policies
 - Providers
 - View
 - bootstrap
 - confia
 - OUTLINE
 - TIMELINE

web.php | BookmarkController.php | Applicant.php | Show.blade.php

app > Models > Applicant.php > Applicant > \$fillable

```
8
9  class Applicant extends Model
10 {
11     use HasFactory;
12
13     protected $fillable = [
14         'job_id',
15         'user_id',
16         'full_name',
17         'contact_phone',
18         'contact_email',
19         'message',
20         'location',
21         'resume_path',
22     ];
23
24     public function job(): BelongsTo
25     {
26         return $this->belongsTo(Job::class);
27     }
28
29     public function user(): BelongsTo
30     {
31         return $this->belongsTo(User::class);
32     }
33 }
```

I build the model. Note that a user can only apply to each job once so the relationship is one to one

EXPLORER

WORKOPIA

app

Http

Controllers

BookmarkController.php

Controller.php

DashboardController.php

HomeController.php

JobController.php

LoginController.php

ProfileController.php

RegisterController.php

Middleware

Models

Applicant.php

Job.php

User.php

Policies

Providers

View

bootstrap

confia

OUTLINE

TIMELINE

web.php

BookmarkController.php

Applicant.php

Show.blade.php

JobContr

app > Models > Applicant.php > ...

6

7

8

9

10

11

12

13

14

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

use Illuminate\Database\Eloquent\Relations\BelongsTo;

use Illuminate\Database\Eloquent\Relations\HasMany;

use Illuminate\Database\Eloquent\Model;

class Applicant extends Model

{

use HasFactory;

protected \$fillable = [...

];

public function job(): BelongsTo

{

return \$this->belongsTo(Job::class);

}

public function user(): BelongsTo

{

return \$this->belongsTo(User::class);

}

public function applicants(): HasMany

{

return \$this->hasMany(Applicant::class);

}

}

Bring in eloquent relation classes

Now The relationship between job and applicant has to be defined. A job can have many applicants so it is a one to many.

🔍 📄 🏠 🌐

EXPLORER

WORKOPIA

app

Http

Controllers

BookmarkController.php

Controller.php

DashboardController.php

HomeController.php

JobController.php

LoginController.php

ProfileController.php

RegisterController.php

Middleware

Models

Applicant.php

Job.php

User.php

Policies

Providers

View

bootstrap

app

BookmarkController.php

Applicant.php

Show.blade.php

JobController.php

header.blade.php

app > Models > User.php > User > applications

12 class User extends Authenticatable

50 }

51

52 // Relate job listing to user

53 public function joblistings(): HasMany

54 {

55 | return \$this->hasMany(Job::class);

56 }

57

58 // Relate User to Job Listing

59 public function bookmarkedJobs(): BelongsToMany

60 {

61 | return \$this->belongsToMany(Job::class, 'job_user_bookmarks')->withTimestamps();

62 }

63

64 // Relation between User and Applicant

65 public function applications(): HasMany

66 {

67 | return \$this->hasMany(Applicant::class, 'user_id');

68 }

69 }

70 }

Now The relationship between user and applicant has to be defined. A user can have many applications so it is a one to many.

12.3 Applicant Form Modal With Alpine.js

```
<!-- Applicant Form -->
<div x-data="{ open: false }" id="applicant-form">
  <button @click="open = true" class="block w-full text-center px-5 py-2.5 mt-5
shadow-sm rounded border text-base font-medium cursor-pointer text-indigo-700 bg-indigo-100
hover:bg-indigo-200">Apply Now</button>

  <div x-show="open" class="fixed inset-0 flex items-center justify-center bg-gray-
900 bg-opacity-50">
    <div @click.away="open = false" class="bg-white p-6 rounded-lg shadow-md w-full
max-w-md">
      <h3 class="text-lg font-semibold mb-4">Apply for {{ $job->title }}</h3>

      <form enctype="multipart/form-data">
        <x-inputs.text id="full_name" name="full_name" label="Full Name"
:required="true" />
        <x-inputs.text id="contact_phone" name="contact_phone" label="Contact
Phone" />
        <x-inputs.text id="contact_email" name="contact_email" label="Contact
Email" :required="true" />
        <x-inputs.text-area id="message" name="message" label="Message" />
        <x-inputs.text id="location" name="location" label="Location" />
        <x-inputs.file id="resume" name="resume" label="Upload Your Resume (pdf)"
:required="true" />
        <button type="submit" class="bg-blue-500 hover:bg-blue-600 text-white px-4
py-2 rounded-md">Submit Application</button>
        <button type="button" @click="open = false" class="ml-2 bg-gray-300
hover:bg-gray-400 text-black px-4 py-2 rounded-md">Cancel</button>
      </form>
    </div>
  </div>
</div>
```

A button will open a modal form with reused input blades.

```

@props(['id', 'name', 'label' => null, 'required' => false])

<div class="mb-4">
  @if($label)
    <label class="block text-gray-700" for="{{ $id }}">{{ $label }}</label>
  @endif
  <input
    id="{{ $id }}"
    type="file"
    name="{{ $name }}"
    class="w-full px-4 py-2 border rounded focus:outline-none @error($name) border-red-500
  @enderror"
    {{ $required ? 'required' : '' }}
  />
  @error($name)
    <p class="text-red-500 text-sm mt-1">{{ $message }}</p>
  @enderror
</div>

```

The file and text input blade components need an extra @prop of required with a default value of false

This is a workaround because during the validate inputs method the modal disappears and the required prop is a means of preventing this.

The modal form looks like this

Apply for Web Developer

Full Name

Contact Phone

Contact Email

Message

Location

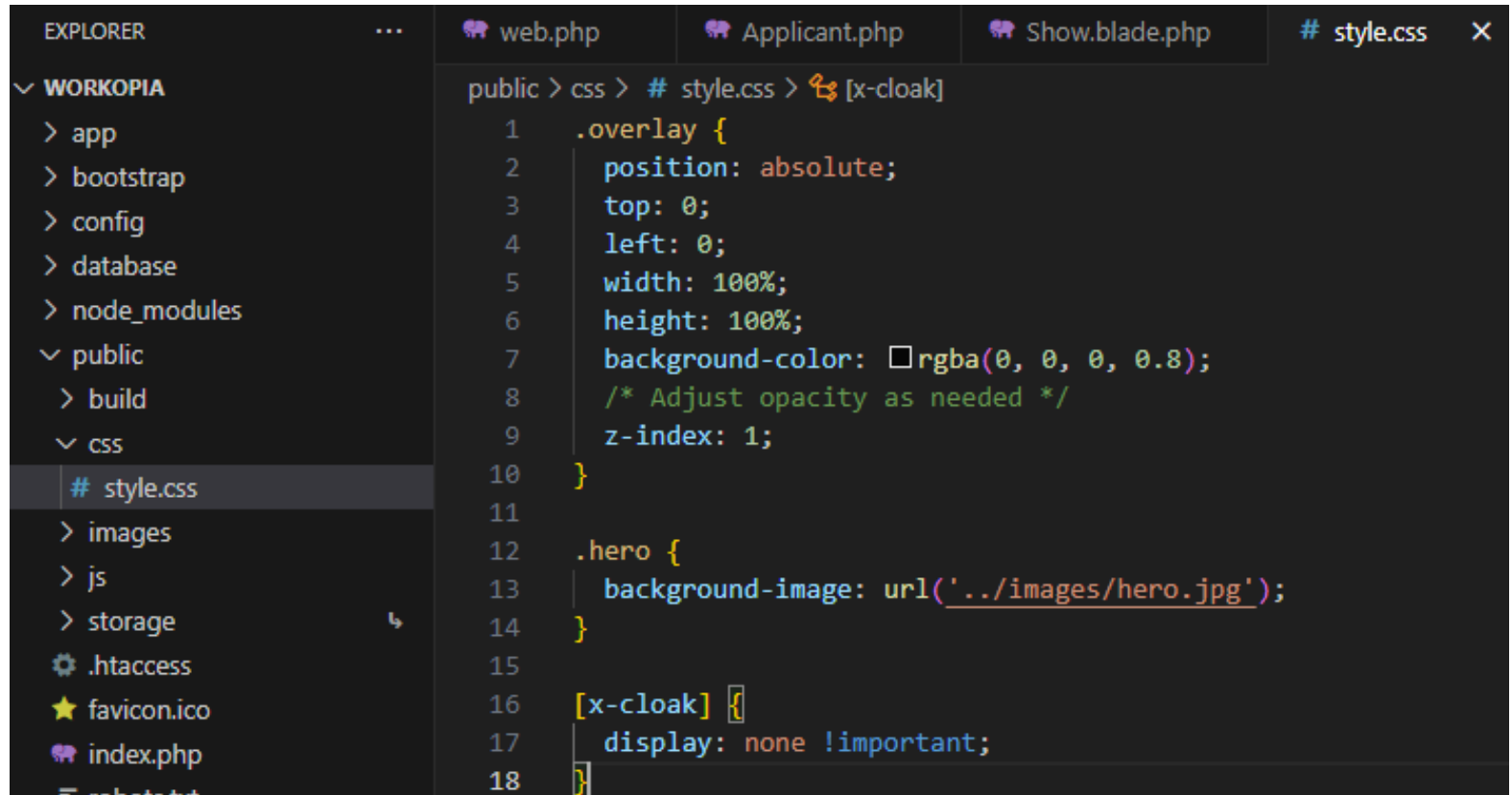
Upload Your Resume (pdf)

Submit Application

Cancel

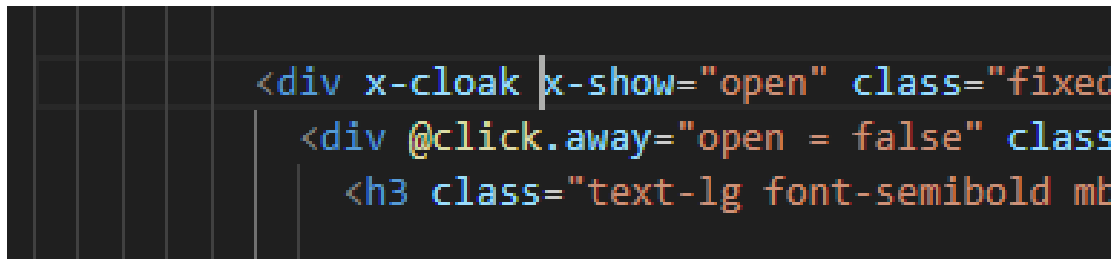
12.4 Fix Modal Blip with x-cloak

<https://alpinejs.dev/directives/cloak>



```
EXPLORER
WORKOPIA
  app
  bootstrap
  config
  database
  node_modules
  public
    build
    css
      # style.css
    images
    js
    storage
  .htaccess
  favicon.ico
  index.php
  robots.txt

public > css > # style.css > [x-cloak]
1  .overlay {
2    position: absolute;
3    top: 0;
4    left: 0;
5    width: 100%;
6    height: 100%;
7    background-color: rgba(0, 0, 0, 0.8);
8    /* Adjust opacity as needed */
9    z-index: 1;
10 }
11
12 .hero {
13   background-image: url('../images/hero.jpg');
14 }
15
16 [x-cloak] {
17   display: none !important;
18 }
```



```
<div x-cloak x-show="open" class="fixed
  <div @click.away="open = false" class
    <h3 class="text-lg font-semibold mb
```

This x-cloak path stops the modal from showing before the page has fully loaded.

12.5 Applicant Controller & Store Method

EXPLORER

WORKOPIA

- database
- node_modules
- public
- resources
- routes
 - console.php
 - web.php 1
- storage
- tests
- vendor
- .editorconfig
- .env
- .env.example
- .gitattributes
- .gitignore
- .rnd
- artisan
- composer.json
- composer.lock
- package-lock.json
- package.json

OUTLINE

web.php 1 X

Applicant.php

Show.blade.php

style.css

JobController.php

header.blade.php

Job.php

routes > web.php > ...

```
8 use App\Http\Controllers\DashboardController;
9 use App\Http\Controllers\ProfileController;
10 use App\Http\Controllers\BookmarkController;
11 use App\Http\Controllers\ApplicantController;
12
13 Route::get('/', [HomeController::class, 'index'])->name('home');
14 // Route::resource('jobs', JobController::class);
15 Route::resource('jobs', JobController::class)->middleware('auth')->only(['create', 'edit', 'update', 'destroy']);
16 Route::resource('jobs', JobController::class)->except(['create', 'edit', 'update', 'destroy']);
17
18 > Route::middleware('guest')->group(function () { ...
23 });
24
25 Route::post('/logout', [LoginController::class, 'logout'])->name('logout');
26
27 Route::get('/dashboard', [DashboardController::class, 'index'])->name('dashboard.index')->middleware('auth');
28
29 Route::put('/profile', [ProfileController::class, 'update'])->name('profile.update')->middleware('auth');
30
31 > Route::middleware('auth')->group(function () { ...
35 });
36
37 Route::post('/jobs/{job}/apply', [ApplicantController::class, 'store'])->name('applicants.store');
38
```

Bring in the ApplicantController

Create the route

Vscode I throwing an error because I have not created the ApplicantController yet!

php artisan make:controller ApplicantController

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\Job;
use App\Models\Applicant;
use Illuminate\Http\RedirectResponse;

class ApplicantController extends Controller
{
    // @desc    Store a new job application
    // @route    POST /jobs/{job}/apply
    public function store(Request $request, Job $job): RedirectResponse
    {
        // Validate the incoming request data
        $validatedData = $request->validate([
            'full_name' => 'required|string|max:255',
            'contact_number' => 'string|max:20',
            'contact_email' => 'required|email',
            'message' => 'string',
            'location' => 'string|max:255',
            'resume' => 'required|file|mimes:pdf|max:4096',
        ]);

        // Handle the resume file upload
        if ($request->hasFile('resume')) {
            $path = $request->file('resume')->store('resumes', 'public');
            $validatedData['resume_path'] = $path;
        }
    }
}
```

The applicant controller is similar to the other controllers in that I validate data, handle the file upload then save to the table.

```

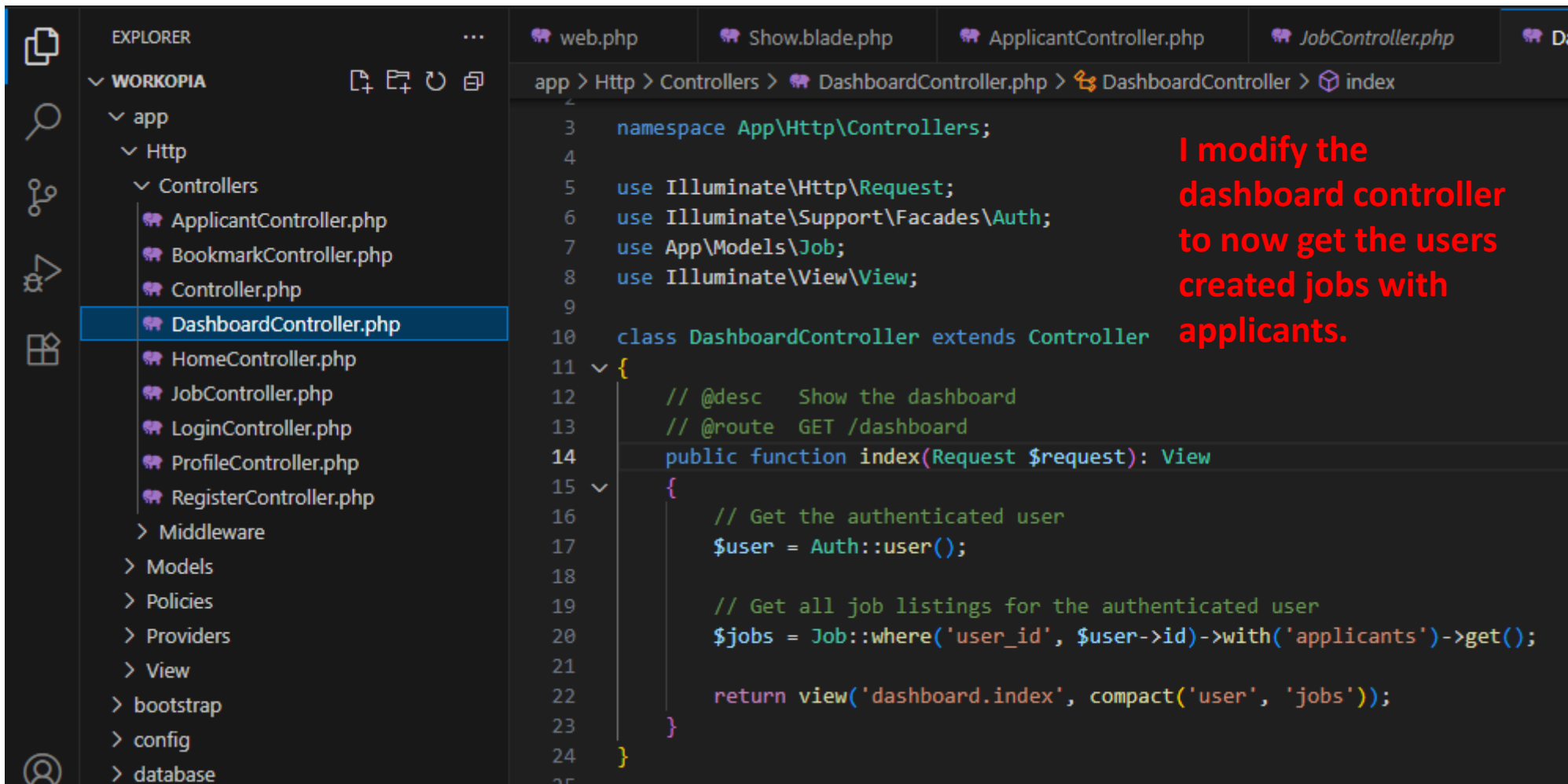
    // Store the application
    $application = new Applicant($validatedData);
    $application->job_id = $job->id;
    /** @disregard p1013 Undefined method */
    $application->user_id = auth()->id();
    $application->save();

    return redirect()->back()->with('success', 'Your application has been submitted!');
}
}

```

| | id
[PK] bigint | user_id
bigint | job_id
bigint | full_name
character varying (255) | contact_phone
character varying (255) | contact_email
character varying (255) | message
text | location
character v |
|---|-------------------|-------------------|------------------|--------------------------------------|--|--|--------------------|-------------------------|
| 1 | 1 | 1 | 8 | Timon Caldwell | [null] | qavo@mailinator.com | Culpa sint molesti | Ullam dolo |

12.6 Show Applicants to owner



EXPLORER

- WORKOPIA
 - app
 - Http
 - Controllers
 - ApplicantController.php
 - BookmarkController.php
 - Controller.php
 - DashboardController.php**
 - HomeController.php
 - JobController.php
 - LoginController.php
 - ProfileController.php
 - RegisterController.php
 - Middleware
 - Models
 - Policies
 - Providers
 - View
 - bootstrap
 - config
 - database

web.php Show.blade.php ApplicantController.php JobController.php

app > Http > Controllers > DashboardController.php > DashboardController > index

```
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use Illuminate\Support\Facades\Auth;
7 use App\Models\Job;
8 use Illuminate\View\View;
9
10 class DashboardController extends Controller
11 {
12     // @desc Show the dashboard
13     // @route GET /dashboard
14     public function index(Request $request): View
15     {
16         // Get the authenticated user
17         $user = Auth::user();
18
19         // Get all job listings for the authenticated user
20         $jobs = Job::where('user_id', $user->id)->with('applicants')->get();
21
22         return view('dashboard.index', compact('user', 'jobs'));
23     }
24 }
```

I modify the dashboard controller to now get the users created jobs with applicants.

EXPLORER

WORKOPIA

resources

views

auth

components

dashboard

index.blade.php

jobs

bookmarked.blade.php

create.blade.php

edit.blade.php

index.blade.php

Show.blade.php

pages

vendor

layout.blade.php

welcome.blade.php

routes

console.php

web.php

storage

OUTLINE

web.php

index.blade.php X

resources > views > dashboard > index.blade.php > x-layout > section.flex.flex-col.md:flex-row.gap-6 > div.bg-white.p-8.rounded-lg.shadow-md

1

<x-layout>

3

<section class="flex flex-col md:flex-row gap-6">

32

<div class="bg-white p-8 rounded-lg shadow-md w-full">

48

</div>

49

50

{{-- Applicants --}}

51

<div class="mt-4">

52

<h4 class="text-lg font-semibold mb-2">Applicants</h4>

53

@forelse(\$job->applicants as \$applicant)

54

<div class="py-2">

55

<p class="text-gray-800">Name: {{ \$applicant->full_name }}</p>

56

<p class="text-gray-800">Phone: {{ \$applicant->contact_phone }}</p>

57

<p class="text-gray-800">Email: {{ \$applicant->contact_email }}</p>

58

<p class="text-gray-800">Message: {{ \$applicant->message }}</p>

59

<p class="text-gray-800 my-4">

60

resume_path) }}" class="text-blue-500 hover:underline" download

61

<i class="fas fa-download"></i> Download Resume

62

63

</p>

64

</div>

65

@empty

66

<p class="text-gray-700">No Applicants</p>

67

@endforelse

68

</div>

69

On the dashboard index.blade, I add a container to show the applicants fields.



Profile Info



Name

Test User

Email

test@test.com

Profile Avatar

Choose File No file chosen

Save

My Job Listings

Software Engineer

Part-Time

Edit

Delete

Applicants

Name: Mante Alexys

Phone:

Email: mante.alexys@example.org

Message: ggg

[Download Resume](#)

Marketing Specialist

Full-Time

Edit

Delete

Applicants

No Applicants

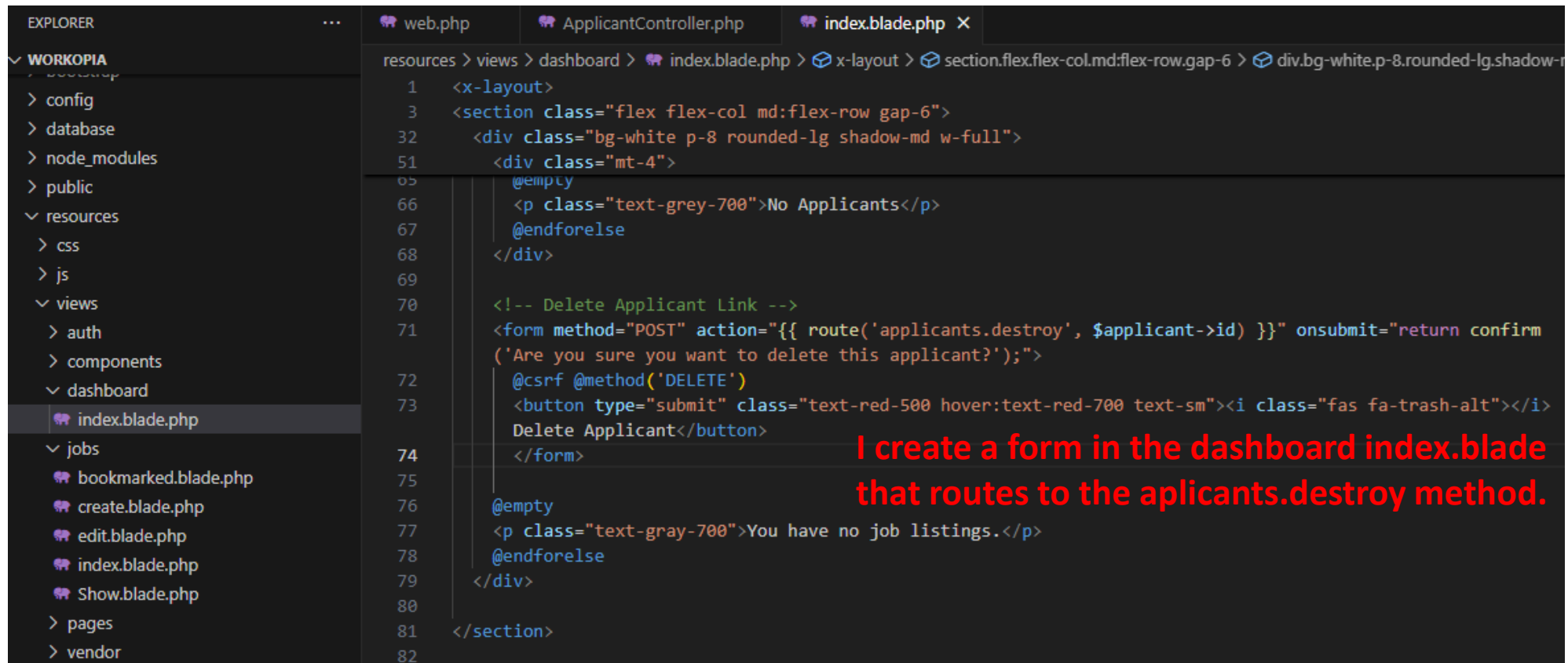
If I log in as another user and apply for one of the jobs that test user owns, then I see the applicant details in the dashboard of test user.

Looking to hire?

Post your job listing now and find the perfect candidate

Create Job

12.7 Delete Applicants



```
EXPLORER
WORKOPIA
  > config
  > database
  > node_modules
  > public
  > resources
    > css
    > js
    > views
      > auth
      > components
      > dashboard
        index.blade.php
      > jobs
        bookmarked.blade.php
        create.blade.php
        edit.blade.php
        index.blade.php
        Show.blade.php
      > pages
      > vendor

resources > views > dashboard > index.blade.php > x-layout > section.flex.flex-col.md:flex-row.gap-6 > div.bg-white.p-8.rounded-lg.shadow-r
1  <x-layout>
3  <section class="flex flex-col md:flex-row gap-6">
32 <div class="bg-white p-8 rounded-lg shadow-md w-full">
51   <div class="mt-4">
65     @empty
66     <p class="text-grey-700">No Applicants</p>
67   @endforelse
68   </div>
69
70   <!-- Delete Applicant Link -->
71   <form method="POST" action="{{ route('applicants.destroy', $applicant->id) }}" onsubmit="return confirm
('Are you sure you want to delete this applicant?');">
72     @csrf @method('DELETE')
73     <button type="submit" class="text-red-500 hover:text-red-700 text-sm"><i class="fas fa-trash-alt"></i>
74     Delete Applicant</button>
75   </form>
76   @empty
77   <p class="text-gray-700">You have no job listings.</p>
78   @endforelse
79   </div>
80
81 </section>
82
```

I create a form in the dashboard index.blade that routes to the applicants.destroy method.


```
EXPLORER    ...    web.php    ApplicantController.php X    index.blade.php

WORKOPIA
├── app
│   ├── Http
│   │   └── Controllers
│   │       ├── ApplicantControll...
│   │       ├── BookmarkControl...
│   │       ├── Controller.php
│   │       ├── DashboardContr...
│   │       ├── HomeController....
│   │       ├── JobController.php
│   │       ├── LoginController.p...
│   │       ├── ProfileController....
│   │       ├── RegisterControlle...
│   └── Middleware
└── ...

app > Http > Controllers > ApplicantController.php > ...
11  class ApplicantController extends Controller
15      public function store(Request $request, Job $job): RedirectResponse
41  }
42
43      // @desc   Delete a job application
44      // @route   DELETE /applicants/{applicant}
45      public function destroy($id): RedirectResponse
46      {
47          $applicant = Applicant::findOrFail($id);
48          $applicant->delete();
49
50          return redirect()->route('dashboard')->with('success', 'Applicant deleted successfully.');
```

The destroy method takes in the id and performs a find or fail (which will throw an error if it does not find it) in this id then deletes it returning to the dashboard with a success message.

12.8 Prevent Multiple Applications

```
public function store(Request $request, Job $job): RedirectResponse
{
    // Check if the user has already applied for the job
    $existingApplication = Applicant::where('job_id', $job->id)
                                    ->where('user_id', auth()->id())
                                    ->exists();

    if ($existingApplication) {
        return redirect()->back()->with('status', 'You have already applied to this job.');
```

I have modified the store method for applicant to include a check to see if the user has already applied for the job

```
    }

    // Validate incoming data
    $validatedData = $request->validate([
        'full_name' => 'required|string',
        'contact_phone' => 'string',
        'contact_email' => 'required|string|email',
        'message' => 'string',
        'location' => 'string',
        'resume' => 'required|file|mimes:pdf|max:2048',
    ]);

    // Handle resume upload
    if ($request->hasFile('resume')) {
        $path = $request->file('resume')->store('resumes', 'public');
        $validatedData['resume_path'] = $path;
    }

    // Store the application
    $application = new Applicant($validatedData);
    $application->job_id = $job->id;
    $application->user_id = auth()->id();
    $application->save();

    return redirect()->back()->with('success', 'Your application has been submitted.');
```

13. Job Search, Maps & Emails

[13.1 Intro](#)

[13.2 Search Component & Route](#)

[13.3 Search Functionality](#)

[13.4 Mapbox Setup](#)

[13.5 Hide Mapbox key](#)

[13.6 Send Emails with Mailers & Mialtrap](#)

[13.7 Sending Data in Emails](#)

[13.8 Email Attachments](#)

[13.9 Setup Emails For Production](#)

13.1 Intro



Laravel From Scratch

Job Search, Maps & Emails - Section Intro

- ✓ **Search Component & Route**
- ✓ **Mapbox Setup**
- ✓ **Geocode Location**
- ✓ **Hide Mapbox Key**
- ✓ **Using Mailers to Send Emails**
- ✓ **Mailtrap SMTP Setup**
- ✓ **Send Data in Emails**
- ✓ **Send Attachments**

13.2 Search Component & Route

```
php artisan make:component Search
```

```
@props(['title' => 'Find Your Dream Job'])
```

Build the Hero blade with the search form

```
<section
  class="hero relative bg-cover bg-center bg-no-repeat h-72 flex items-center"
>
  <div class="overlay"></div>
  <div class="container mx-auto text-center z-10">
    <h2 class="text-4xl text-white font-bold mb-4">{{ $title }}</h2>
    <form class="block mx-5 md:mx-auto md:space-x-2" method="GET" action="{{
route('jobs.search') }}">
      <input type="text" name="keywords" placeholder="Keywords" class="w-full md:w-72 bg-
white px-4 py-3 focus:outline-none" value="{{ request('keywords') }}" />
      <input type="text" name="location" placeholder="Location" class="w-full md:w-72 bg-
white px-4 py-3 focus:outline-none" value="{{ request('location') }}" />
      <button
        class="w-full md:w-auto bg-blue-700 hover:bg-blue-600 text-white px-4 py-3
focus:outline-none"
      >
        <i class="fa fa-search mr-1"></i> Search
      </button>
    </form>

    <!-- Back Button -->
    @if(request()->has('keywords') || request()->has('location'))
      <a href="{{ route('jobs.index') }}" class="bg-gray-700 hover:bg-gray-600 text-white px-4
py-2 rounded mb-4 inline-block"><i class="fa fa-arrow-left mr-1"></i> Back</a>
    @endif

  </div>
</section>
```

EXPLORER

WORKOPIA

resources

views

components

dashboard

jobs

pages

vendor

layout.blade.php

welcome.blade.php

routes

console.php

web.php

storage

tests

vendor

.editorconfig

.env

.env.example

.gitattributes

.gitignore

red

web.php

ApplicantController.php

JobController.php

hero.blade.php

index.blade.php

routes > web.php > ...

1 <?php

2

3 use Illuminate\Support\Facades\Route;

4 use App\Http\Controllers\JobController;

5 use App\Http\Controllers\HomeController;

6 use App\Http\Controllers\LoginController;

7 use App\Http\Controllers\RegisterController;

8 use App\Http\Controllers\DashboardController;

9 use App\Http\Controllers\ProfileController;

10 use App\Http\Controllers\BookmarkController;

11 use App\Http\Controllers\ApplicantController;

12

13 Route::get('/', [HomeController::class, 'index'])->name('home');

14 Route::get('/jobs/search', [JobController::class, 'search'])->name('jobs.search');

15 Route::resource('jobs', JobController::class)->middleware('auth')->only(['create', 'edit', 'update', 'destroy']);

16 Route::resource('jobs', JobController::class)->except(['create', 'edit', 'update', 'destroy']);

17

18 Route::middleware('guest')->group(function () {

19 Route::get('/register', [RegisterController::class, 'register'])->name('register');

20 Route::post('/register', [RegisterController::class, 'store'])->name('register.store');

21 Route::get('/login', [LoginController::class, 'login'])->name('login');

22 Route::post('/login', [LoginController::class, 'authenticate'])->name('login.authenticate');

23 });

Add the route to /jobs/search

13.3 Search Functionality

```
// @desc    Search for jobs
// @route   GET /jobs/search
public function search(Request $request)
{
    $keywords = strtolower($request->input('keywords'));
    $location = strtolower($request->input('location'));

    $query = Job::query();
    if ($keywords) {
        $query->where(function ($q) use ($keywords) {
            $q->whereRaw('LOWER(title) like ?', ['%' . $keywords . '%'])
                ->orWhereRaw('LOWER(description) like ?', ['%' . $keywords . '%']);
        });
    }

    if ($location) {
        $query->where(function ($q) use ($location) {
            $q->whereRaw('LOWER(address) like ?', ['%' . $location . '%'])
                ->orWhereRaw('LOWER(city) like ?', ['%' . $location . '%'])
                ->orWhereRaw('LOWER(state) like ?', ['%' . $location . '%'])
                ->orWhereRaw('LOWER(zipcode) like ?', ['%' . $location . '%']);
        });
    }

    $jobs = $query->paginate(9);

    return view('jobs.index')->with('jobs', $jobs);
}
```

Assign the q values to variables & lowercase all letters

Use a query builder by creating a query variable and set it to Job::query()

```
public function search(Request $request)
```

```
{
```

```
    $keywords = strtolower($request->input('keywords'));  
    $location = strtolower($request->input('location'));
```

```
    $query = Job::query();
```

```
    if ($keywords) {
```

```
        $query->where(function ($q) use ($keywords) {
```

```
            $q->whereRaw('LOWER(title) like ?', ['%' . $keywords . '%'])
```

```
                ->orWhereRaw('LOWER(description) like ?', ['%' . $keywords . '%']);
```

```
        });
```

```
    }
```

How the query logic works

This is where the main query logic happens. The `where()` method is being called on the query builder object (`\$query`).

Inside the `where()` method, a closure (anonymous function) is used. This closure allows me to encapsulate multiple conditions for the where clause.

`use (\$keywords)` is used to pass variables from the parent scope (in this case, `\$keywords`) into the closure. Without `use`, the closure wouldn't have access to the `\$keywords` variable since it's defined outside of the closure.

I am using the `whereRaw` method to pass in raw SQL so that I can use LOWER to make the search case-insensitive.


```

<x-layout>
  <x-slot name="title">All Jobs</x-slot>
  <h1 class="text-center text-3xl mb-4 font-bold border border-gray-300 p-3">All Jobs</h1>

  <div class="grid grid-cols-1 md:grid-cols-3 gap-4 mb-6">
    @forelse($jobs as $job)
      <x-job-card :job="$job" />
    @empty
      <p>No jobs found</p>
    @endforelse
  </div>

  <!-- Back Button -->
  @if(request()->has('keywords') || request()->has('location'))
    <a href="{{ route('home') }}" class="bg-gray-700 hover:bg-gray-600 text-white px-4 py-2
rounded mb-4 inline-block"><i class="fa fa-arrow-left mr-1"></i> Back</a>
  @endif

  <!-- Pagination Links -->
  <div class="mt-4">{{ $jobs->links() }}</div>

</x-layout>

```

The back button returns to the home page of there are search q values in the request object

13.4 Mapbox Setup

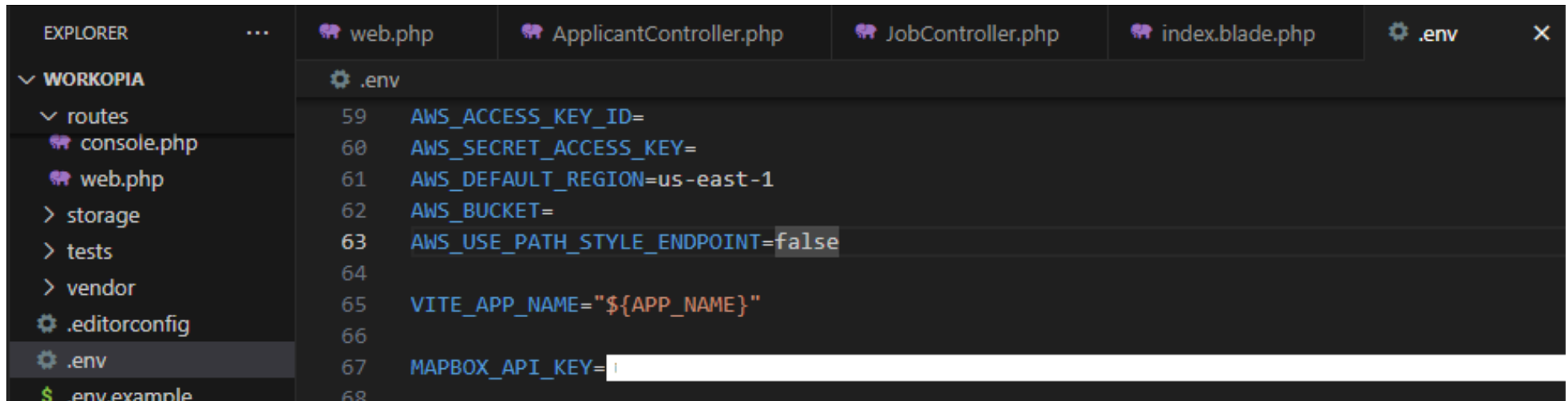
Create a Mapbox account. Go to [Mapbox](#) and sign up for a free account. Go to [Mapbox Account](#) and copy the API key (called a public token).



The screenshot shows the Mapbox Admin console. On the left is a sidebar with 'Admin' and sub-items: 'Tokens', 'Invoices', 'Statistics', and 'Settings'. The main area displays a table of tokens. The first row is for the 'Default public token', which is redacted with a black box. It was created '3 minutes ago' and has 'N/A' URLs. A 'Refresh' button is visible next to the token.

| Name | Token | Last modified | URLs |
|--|------------|---------------|------|
| Default public token
<small>Use protected tokens for live applications. See our token management best practices page to learn more.</small> | [Redacted] | 3 minutes ago | N/A |

Add the Mapbox public token to the .env environment variables.



The screenshot shows a code editor with a sidebar on the left containing 'EXPLORER' and a list of files: 'web.php', 'ApplicantController.php', 'JobController.php', 'index.blade.php', and '.env'. The '.env' file is selected and open in the main editor. The file contains several environment variables, and the 'MAPBOX_API_KEY=' line is being added at the bottom.

```
59 AWS_ACCESS_KEY_ID=  
60 AWS_SECRET_ACCESS_KEY=  
61 AWS_DEFAULT_REGION=us-east-1  
62 AWS_BUCKET=  
63 AWS_USE_PATH_STYLE_ENDPOINT=false  
64  
65 VITE_APP_NAME="${APP_NAME}"  
66  
67 MAPBOX_API_KEY=  
68
```

```
<link href="https://api.mapbox.com/mapbox-gl-js/v2.7.0/mapbox-gl.css" rel="stylesheet"/>
<script src="https://api.mapbox.com/mapbox-gl-js/v2.7.0/mapbox-gl.js"></script>
<script>
    document.addEventListener('DOMContentLoaded', function () {
        // Your Mapbox access token
        mapboxgl.accessToken = "{{ env('MAPBOX_API_KEY') }}";

        // Initialize the map
        const map = new mapboxgl.Map({
            container: 'map', // ID of the container element
            style: 'mapbox://styles/mapbox/streets-v11', // Map style
            center: [-74.5, 40], // Default center
            zoom: 9, // Default zoom level
        });

        // Get address from Laravel view
        const city = '{{ $job->city }}';
        const state = '{{ $job->state }}';
        const address = city + ', ' + state;
```

This code block goes at the bottom of the show blade. It contains the CSS for mapbox plus the script.

After is some custom JS code to load the map once the DOM is initialised.

I am getting the geo location address from the Laravel View

```
// Geocode the address
fetch(
  `https://api.mapbox.com/geocoding/v5/mapbox.places/${encodeURIComponent(
    address
  )}.json?access_token=${mapboxgl.accessToken}`
)
  .then((response) => response.json())
  .then((data) => {
    if (data.features.length > 0) {
      const [longitude, latitude] = data.features[0].center;

      // Center the map and add a marker
      map.setCenter([longitude, latitude]);
      map.setZoom(14);

      new mapboxgl.Marker().setLngLat([longitude, latitude]).addTo(map);
    } else {
      console.error('No results found for the address.');
```

```
}
```

```
})
```

```
.catch((error) => console.error('Error geocoding address:', error));
```

```
});
```

```
</script>
```

EXPLORER

WORKOPIA

- > app
- > bootstrap
- > config
- > database
- > node_modules
- public
 - > build
 - css
 - # style.css
 - > images
 - > js
 - > storage
 - .htaccess
 - ★ favicon.ico
 - 🐼 index.php
 - ☰ robots.txt
 - resources

...

Show.blade.php# style.css

public > css > # style.css > #map

10 }

11

12 .hero {

13 | background-image: url('../images/hero.jpg');

14 }

15

16 [x-cloak] {

17 | display: none !important;

18 }

19

20 #applicant-form {

21 | position: relative;

22 | z-index: 2;

23 }

24

25 #map {

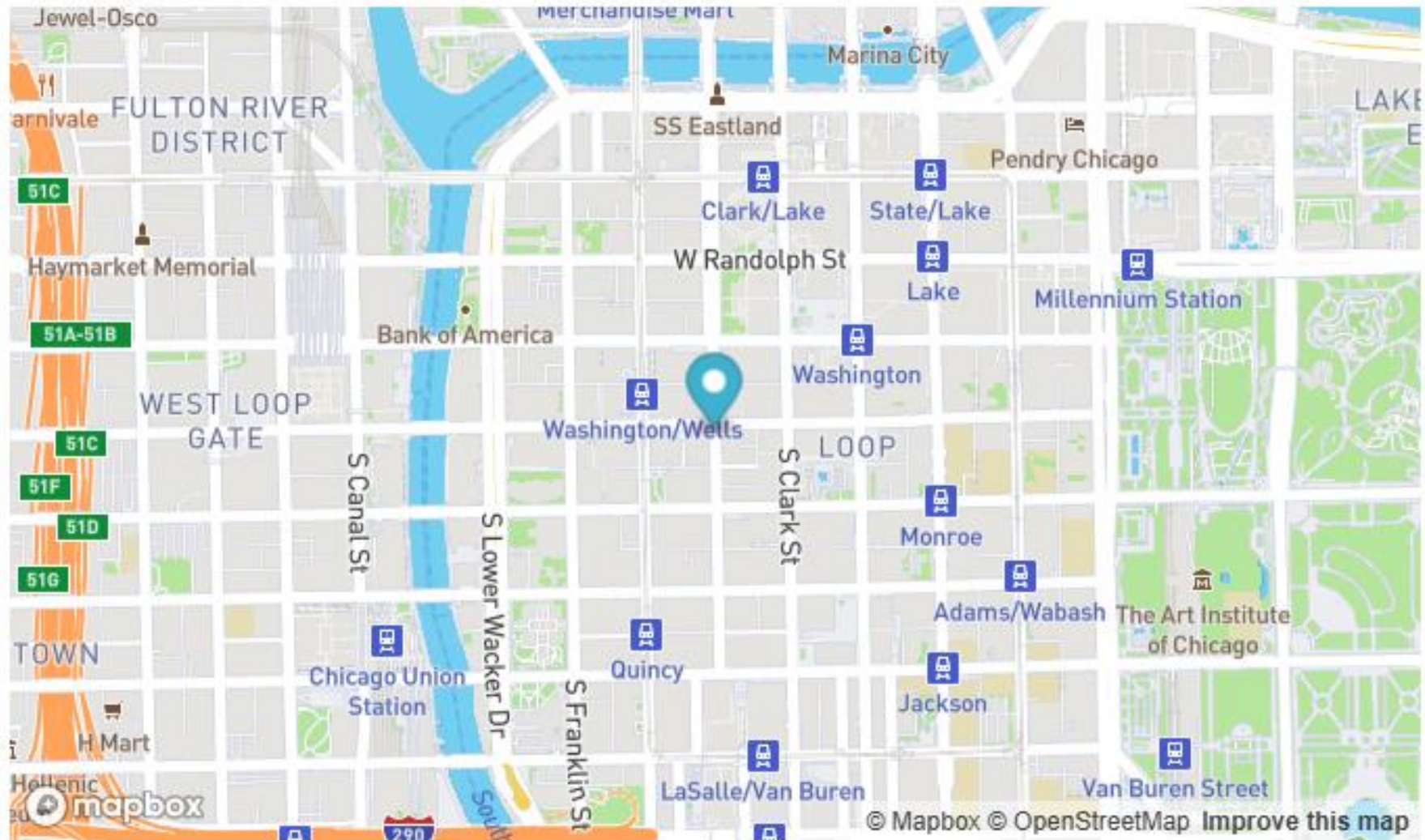
26 | z-index: 1;

27 | height: 400px;

28 }

In public\css\style.css I have added some css so that the applicant form has a higher z-index than the map so that it shows on top

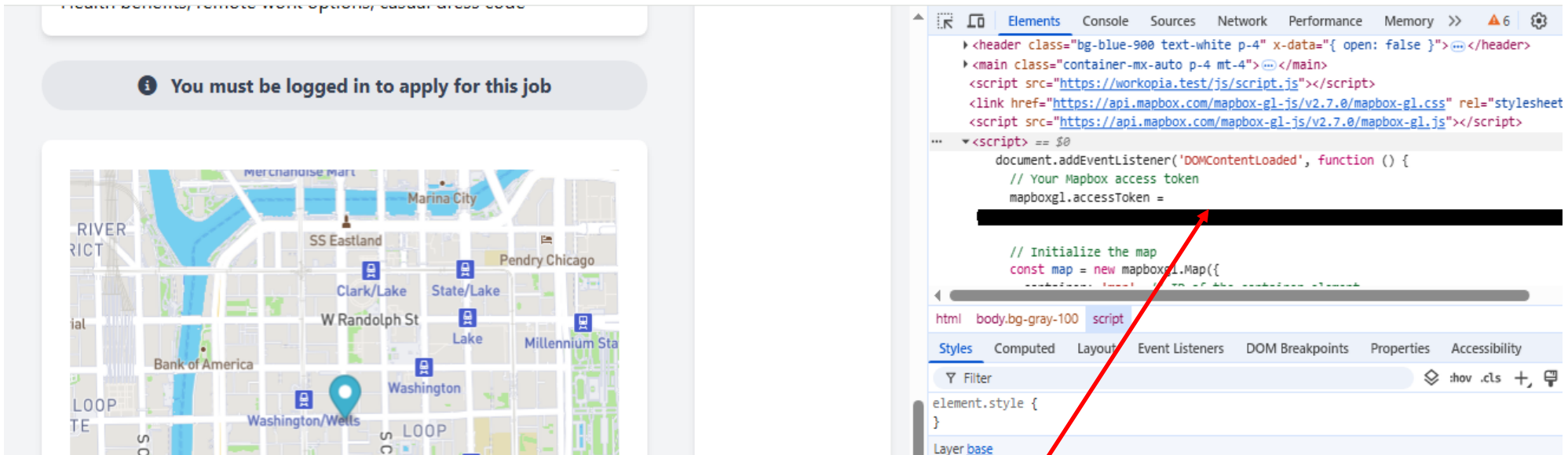
This now loads a map with a POI for the job location in the job listings details page.



13.5 Hide Mapbox Key

Right now, the JS code that renders the map is using the API key/token from the .env file in the code that executes in the user's browser so the key is visible and exposed. Someone can steal it and use it on their own project and leave me with a huge bill.

Let's hide it by using a proxy endpoint. We will create a new route that will call the Mapbox API and return the data to the client.



The image shows a web application interface on the left and its browser developer tools on the right. The web application displays a map of Chicago with labels for 'RIVER RICT', 'LOOP TE', 'SS Eastland', 'Clark/Lake', 'State/Lake', 'W Randolph St', 'Lake', 'Millennium Sta', 'Bank of America', 'Washington', 'Washington/Wells', and 'LOOP'. A notification banner at the top says 'You must be logged in to apply for this job'. The browser developer tools on the right show the 'Elements' tab with the following HTML structure:

```
<header class="bg-blue-900 text-white p-4" x-data="{ open: false }"></header>
<main class="container-mx-auto p-4 mt-4">
  <script src="https://workopia.test/js/script.js"></script>
  <link href="https://api.mapbox.com/mapbox-gl-js/v2.7.0/mapbox-gl.css" rel="stylesheet">
  <script src="https://api.mapbox.com/mapbox-gl-js/v2.7.0/mapbox-gl.js"></script>
  <script == $0
    document.addEventListener('DOMContentLoaded', function () {
      // Your Mapbox access token
      mapboxgl.accessToken =
      // Initialize the map
      const map = new mapboxgl.Map({
        // ...
      })
    })
  </script>
</main>
```

A red arrow points from the text 'Exposed private API key in client-side browser' to the line `mapboxgl.accessToken =` in the script, which is followed by a blacked-out section of code.

Exposed private API key in client-side browser

In the routing file (web.php) I am going to bring in the GeocodeController and add a new route to this controller.

```
use App\Http\Controllers\GeocodeController;
```

```
Route::get('/geocode', [GeocodeController::class, 'geocode']);
```

```
php artisan make:controller GeocodeController
```

EXPLORER

WORKOPIA

- app
 - Http
 - Controllers
 - ApplicantController.php
 - BookmarkController.php
 - Controller.php
 - DashboardController.php
 - GeocodeController.php**
 - HomeController.php
 - JobController.php
 - LoginController.php
 - ProfileController.php
 - RegisterController.php
 - Middleware
 - Models
 - Policies
 - Providers
 - View

app > Http > Controllers > GeocodeController.php > ...

```
1 <?php
2
3 namespace App\Http\Controllers;
4
5 use Illuminate\Http\Request;
6 use Illuminate\Support\Facades\Http;
7
8 class GeocodeController extends Controller
9 {
10     public function geocode(Request $request): array
11     {
12         $address = $request->input('address');
13         $accessToken = env('MAPBOX_API_KEY');
14
15         $response = Http::get("https://api.mapbox.com/geocoding/v5/mapbox.places/{$address}.json", [
16             'access_token' => $accessToken
17         ]);
18
19         return $response->json();
20     }
21 }
```

The GeocodeController calls the API endpoint with the private API key/token ensuring that it is not exposed to the browser because it is executed serverside.

The method returns the JSON object from the API


```
fetch(`/geocode?address=${encodeURIComponent(address)}`)  
  .then(response => response.json())  
  .then(data => {  
    if (data.features.length > 0) {  
      const [longitude, latitude] = data.features[0].center;  
  
      // Center the map and add a marker  
      map.setCenter([longitude, latitude]);  
      map.setZoom(14);  
  
      new mapboxgl.Marker()  
        .setLngLat([longitude, latitude])  
        .addTo(map);  
    } else {  
      console.error('No results found for the address.');    }  
  })  
  .catch(error => console.error('Error geocoding address:', error));  
});
```

**The javascript is modified to now
call the GeoCodeController
method via the endpoint route.**

13.6 Send Emails with Mailers & Mailtrap.io

We can send emails from Laravel. What I would like to do is notify the user via email when someone applies to their job. Using Mailtrap, which is an emailing platform for testing and sending emails in production. We're going to use it for both. There's a very generous free tier that let's us send up to 200 emails per day. Obviously if you're building a production site then you'll want to look into the premium plans, but the free plan is more than enough for this project.

Since we're building locally, we can't just simply send emails from Laravel. Mailtrap gives us a sandbox smtp server to use in development. So we're going to get that setup now.

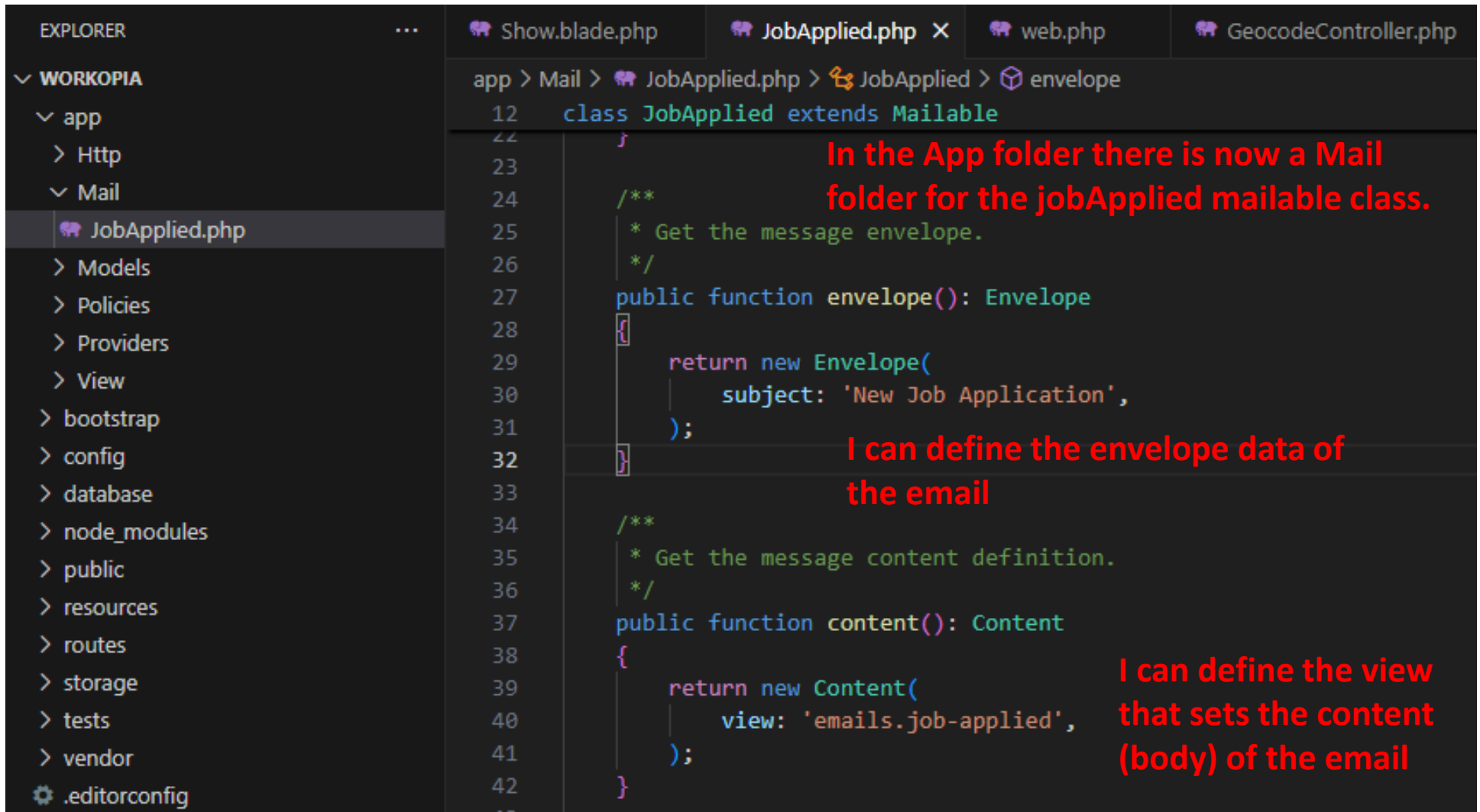
Start by setting up [Mailtrap](#) by signing up for an account. Once you are signed up, click on "Start Testing" under "Email Testing". This will allow us to test emails in our local environment.

Once you do that, click on the "Add Inbox" button and give it a name. I will call it "Workopia". You will then see your inbox. Click on that and where it says "Code Samples", click on PHP and choose "Laravel 9+" from the dropdown. This will give you the configuration that you need to add to your ``.env`` file. It will look something like this:

```
MAIL_MAILER=smtp
MAIL_HOST=sandbox.smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=xxxx
MAIL_PASSWORD=xxx
MAIL_FROM_ADDRESS="noreply@workopia.dev"
MAIL_FROM_NAME="{APP_NAME}"
```

In Laravel, we can send emails using a Mailable. A Mailable is a class that contains the logic for sending an email. We can create a Mailable by running the following command:

```
php artisan make:mail JobApplied
```



The screenshot shows an IDE with the Explorer panel on the left and the Code editor on the right. The Explorer panel shows the project structure with the following folders and files:

- WORKOPIA
 - app
 - Http
 - Mail
 - JobApplied.php
 - Models
 - Policies
 - Providers
 - View
 - bootstrap
 - config
 - database
 - node_modules
 - public
 - resources
 - routes
 - storage
 - tests
 - vendor
 - .editorconfig

The Code editor shows the following PHP code for the JobApplied Mailable class:

```
12 class JobApplied extends Mailable
23 {
24     /**
25      * Get the message envelope.
26      */
27     public function envelope(): Envelope
28     {
29         return new Envelope(
30             subject: 'New Job Application',
31         );
32     }
33
34     /**
35      * Get the message content definition.
36      */
37     public function content(): Content
38     {
39         return new Content(
40             view: 'emails.job-applied',
41         );
42     }
43 }
```

Annotations in red text are present in the code editor:

- In the App folder there is now a Mail folder for the jobApplied mailable class.** (Next to line 23)
- I can define the envelope data of the email** (Next to line 31)
- I can define the view that sets the content (body) of the email** (Next to line 40)

EXPLORER

WORKOPIA

- > config
- > database
- > node_modules
- > public
- > resources
 - > css
 - > js
 - > views
 - > auth
 - > components
 - > dashboard
 - > emails
 - job-applied.blade.php
 - jobs

Show.blade.phpJobApplied.phpjob-applied.blade.php Xweb.php

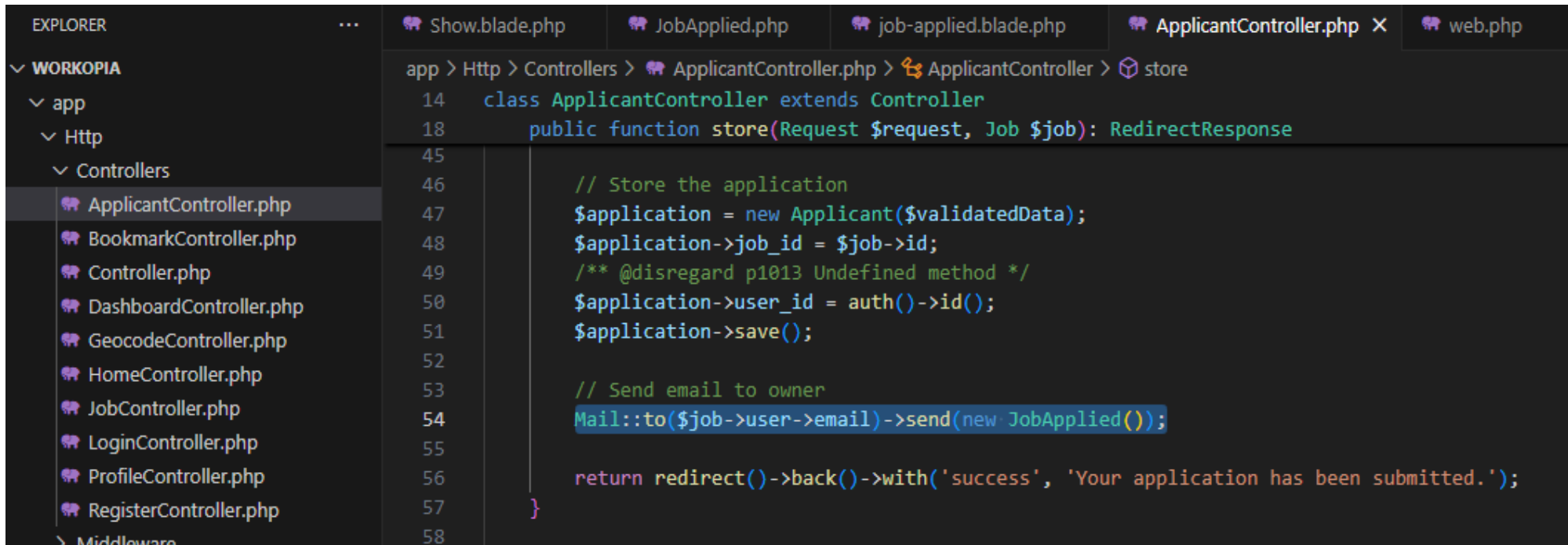
resources > views > emails > job-applied.blade.php > html

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Workopia Job Application</title>
5   </head>
6
7   <body>
8     <p>There has been a new job application to your Workopia listing</p>
9
10    <p>Login to your Workopia account to view the application.</p>
11  </body>
12 </html>
```

I create a file at `resources/views/emails/job-applied.blade.php`.
In this file, I add the HTML that I want to send in the email.

Now that the Mailable is set up, we can send the email. Let's send the email when someone applies to a job. In the `ApplicantController` I bring in the Email façade and the Mail controller for JobApplied.

```
use Illuminate\Support\Facades\Mail;  
use App\Mail\JobApplied;
```

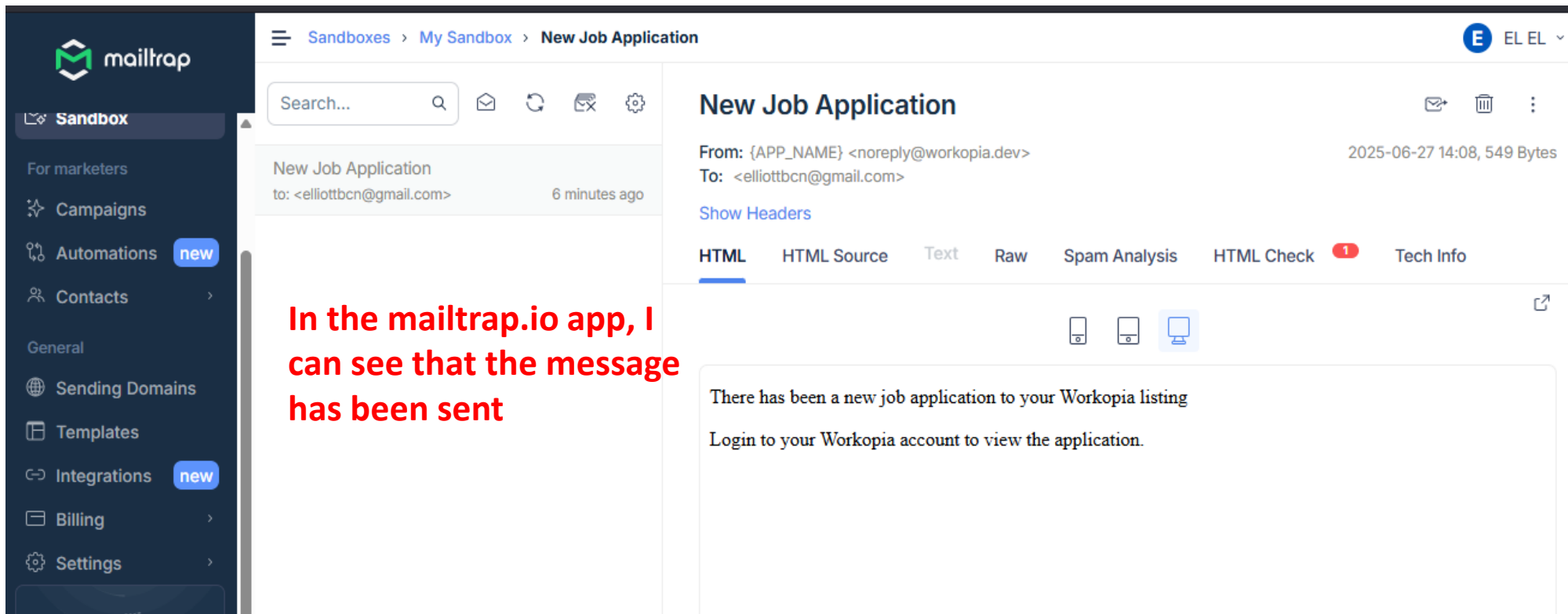


```
app > Http > Controllers > ApplicantController.php > ApplicantController > store  
14 class ApplicantController extends Controller  
18     public function store(Request $request, Job $job): RedirectResponse  
45  
46         // Store the application  
47         $application = new Applicant($validatedData);  
48         $application->job_id = $job->id;  
49         /** @disregard p1013 Undefined method */  
50         $application->user_id = auth()->id();  
51         $application->save();  
52  
53         // Send email to owner  
54         Mail::to($job->user->email)->send(new JobApplied());  
55  
56         return redirect()->back()->with('success', 'Your application has been submitted.');
```

In the applicant controller store method, I can use a mail function to send an email
`Mail::to($job->user->email)->send(new JobApplied());`

I now log into my app and change the test user's email to a real email address that I own. I log out and create a new user with another genuine email address that I own. I apply for one of the jobs of test user.

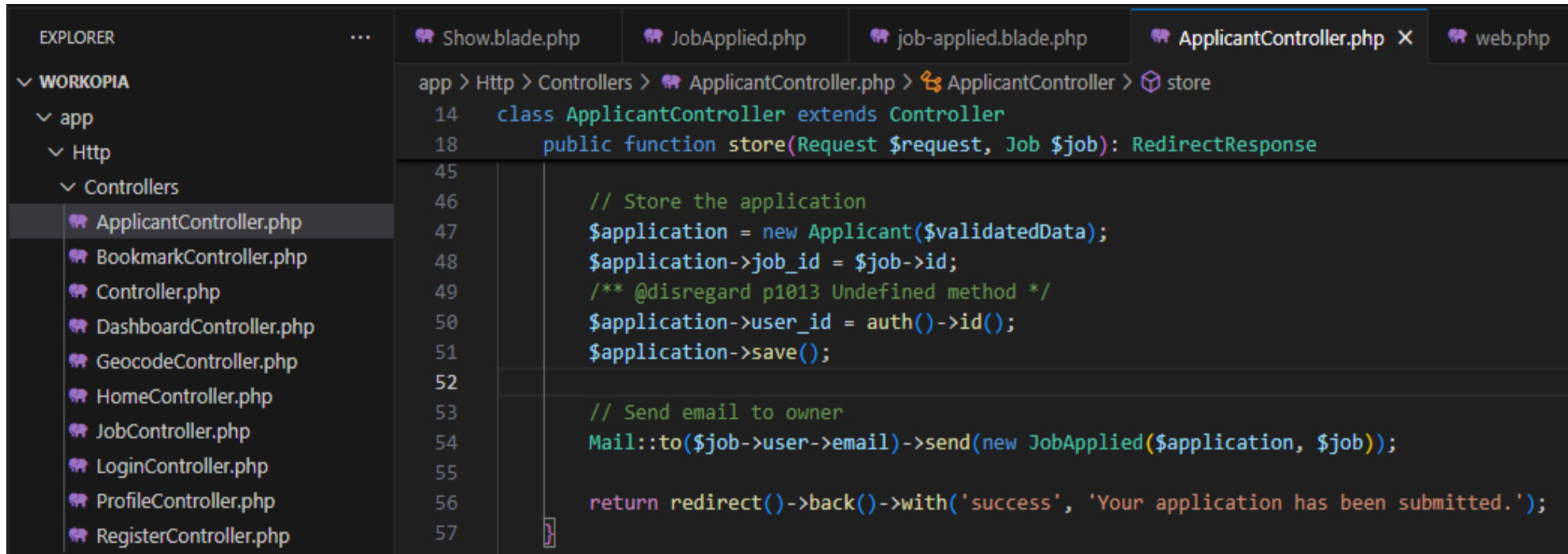
When I log out and log back in as test user, I see the job application.



The screenshot displays the Mailtrap.io web application interface. On the left is a dark sidebar with the Mailtrap logo and a navigation menu including 'Sandbox', 'Campaigns', 'Automations' (marked 'new'), 'Contacts', 'General', 'Sending Domains', 'Templates', 'Integrations' (marked 'new'), 'Billing', and 'Settings'. The main area is titled 'Sandboxes > My Sandbox > New Job Application' and shows an email preview. The email is from '{APP_NAME} <noreply@workopia.dev>' to '<elliottbcn@gmail.com>' with a subject of 'New Job Application' and a timestamp of '6 minutes ago'. The email content, viewed in 'HTML' format, states: 'There has been a new job application to your Workopia listing. Login to your Workopia account to view the application.' A red text overlay on the left side of the email preview reads: 'In the mailtrap.io app, I can see that the message has been sent'. The top right of the interface shows a user profile 'EL EL'.

13.7 Sending Data in Emails

I have the email being sent to the job owner. Let's add some data to the email. I want the job title and all the application data. I can do this by passing in data to the Mailable class.



```
EXPLORER
WORKOPIA
  app
    Http
      Controllers
        ApplicantController.php
        BookmarkController.php
        Controller.php
        DashboardController.php
        GeocodeController.php
        HomeController.php
        JobController.php
        LoginController.php
        ProfileController.php
        RegisterController.php

ApplicantController.php
14 class ApplicantController extends Controller
18     public function store(Request $request, Job $job): RedirectResponse
45
46         // Store the application
47         $application = new Applicant($validatedData);
48         $application->job_id = $job->id;
49         /** @disregard p1013 Undefined method */
50         $application->user_id = auth()->id();
51         $application->save();
52
53         // Send email to owner
54         Mail::to($job->user->email)->send(new JobApplied($application, $job));
55
56         return redirect()->back()->with('success', 'Your application has been submitted.');
```

In the applicant controller store method, I append the mailer to pass in the \$job and \$application objects in the send method

```
Mail::to($job->user->email)->send(new JobApplied($application, $job));
```

EXPLORER

WORKOPIA

- app
 - Http
 - Controllers
 - ApplicantController.php
 - BookmarkController.php
 - Controller.php
 - DashboardController.php
 - GeocodeController.php
 - HomeController.php
 - JobController.php
 - LoginController.php
 - ProfileController.php
 - RegisterController.php
 - Middleware
 - Mail
 - JobApplied.php

app > Mail > JobApplied.php > JobApplied > __construct

```
10 use Illuminate\Queue\SerializesModels;
11
12 class JobApplied extends Mailable
13 {
14     use Queueable, SerializesModels;
15
16     public $application;
17     public $job;
18
19     /**
20      * Create a new message instance.
21      */
22     public function __construct($application, $job)
23     {
24         $this->application = $application;
25         $this->job = $job;
26     }
27 }
```

In the job applied class, I pass in the \$job and \$application object.

And add them into the constructor to be able to use them globally in the class

The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like 'database', 'node_modules', 'public', 'resources', 'views', and 'emails'. The 'emails' folder is expanded, showing a file named 'job-applied.blade.php'. The code editor shows the content of this file, which is a Blade template. The template includes a doctype, html, head, and body tags. The body contains a paragraph of text, a strong tag for 'Job Title', and a series of strong tags for application details (Full Name, Contact Number, Contact Email, Message, Location). The template uses Laravel Blade syntax for embedding PHP variables. The cursor is positioned at the end of the 'Application Details' strong tag on line 12.

```
resources > views > emails > job-applied.blade.php > html > body > p
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Workopia Job Application</title>
5   </head>
6
7   <body>
8     <p>There has been a new job application to your Workopia listing</p>
9
10    <p><strong>Job Title:</strong> {{ $job->title }}</p>
11
12    <p><strong>Application Details:</strong></p>
13
14    <p><strong>Full Name:</strong> {{ $application->full_name }}</p>
15    <p><strong>Contact Number:</strong> {{ $application->contact_number }}</p>
16    <p><strong>Contact Email:</strong> {{ $application->contact_email }}</p>
17    <p><strong>Message:</strong> {{ $application->message }}</p>
18    <p><strong>Location:</strong> {{ $application->location }}</p>
19
20    <p>Login to your Workopia account to view the application.</p>
21  </body>
22 </html>
```

I modify the job-applied.blade to include markup containing properties from the \$job and \$application objects

Search...



New Job Application

to: <elliottbcn@gmail.com>

8 minutes ago

New Job Application



From: workopia.test <noreply@workopia.test>

2025-06-27 14:49, 964 Bytes

To: <elliottbcn@gmail.com>

[Show Headers](#)

HTML

HTML Source

Text

Raw

Spam Analysis

HTML Check

1

Tech Info

In the mailtrap.io app, I
can see that the message
has been sent and now
contains body text



There has been a new job application to your Workopia listing

Job Title: Software Engineer

Application Details:

Full Name: Elliott Farmer

Contact Number:

Contact Email: elliott_w_farmer@yahoo.co.uk

Message: Testing Email Body functionality

Location: Totam cumque aliqua

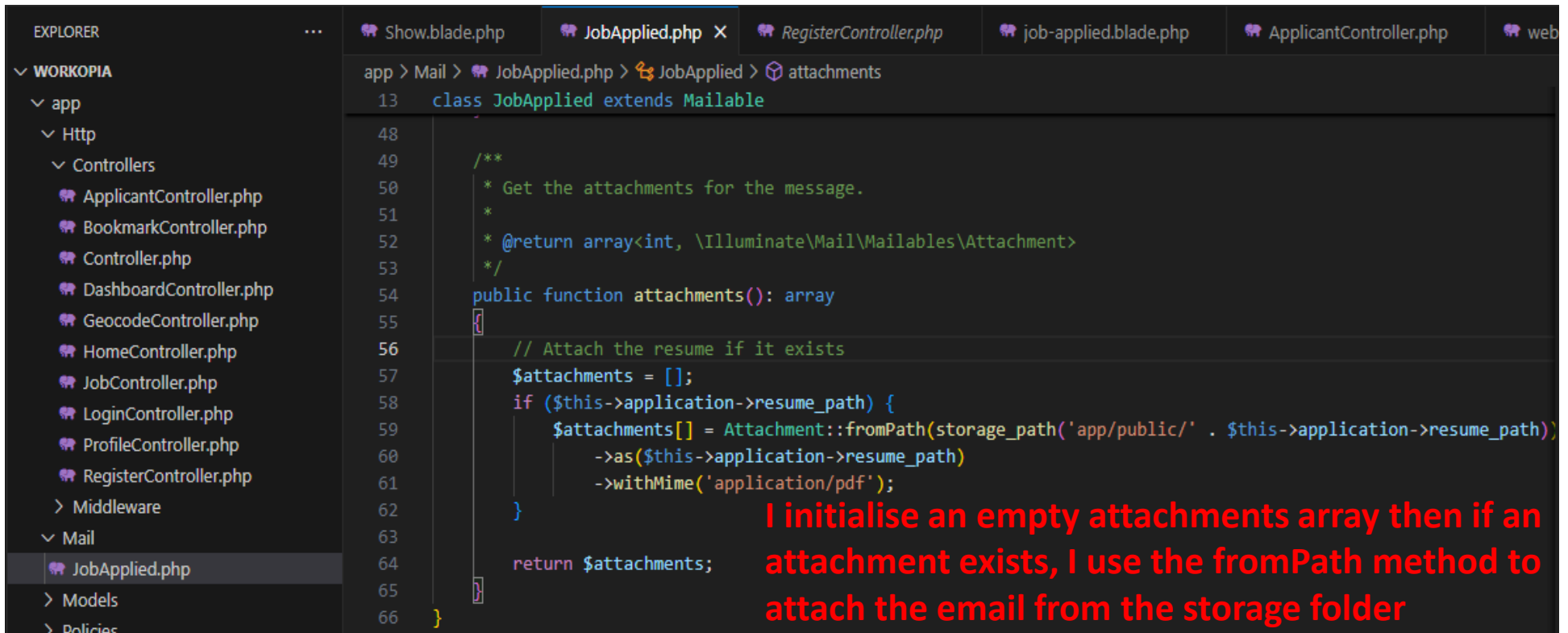
Login to your Workopia account to view the application



13.8 Sending Attachments in Emails

I am already passing the entire `\$application` variable to the email, which includes the `resume\_path` property. I can use this property to attach the resume to the email. In the jobApplied mailable, I need to bring in the Laravel attachment façade.

```
use Illuminate\Mail\Mailables\Attachment;
```



```
13 class JobApplied extends Mailable
48
49 /**
50  * Get the attachments for the message.
51  *
52  * @return array<int, \Illuminate\Mail\Mailables\Attachment>
53  */
54 public function attachments(): array
55 {
56     // Attach the resume if it exists
57     $attachments = [];
58     if ($this->application->resume_path) {
59         $attachments[] = Attachment::fromPath(storage_path('app/public/' . $this->application->resume_path))
60             ->as($this->application->resume_path)
61             ->withMime('application/pdf');
62     }
63
64     return $attachments;
65 }
66 }
```

I initialise an empty attachments array then if an attachment exists, I use the fromPath method to attach the email from the storage folder

13.9 Setup Emails for Production

mailtrap

Sending Domains > New

E EL EL

Set up Your Domain

Connect your domain to Mailtrap to start sending emails. This is usually your company's website. We'll handle the sending and the data management, but the emails are sent from your domain name.

Before adding a domain, please ensure the following:

- All email-triggering forms on your website are protected against bots. [Learn Why](#)
- The .env files containing sensitive data are not publicly accessible.

mydomain.com [Add Your Domain](#)

Migrating from another solution? Don't forget to [add your suppression list](#)

To send emails in a production environment I would need to add a sending domain. This needs to be a real domain and the hosting provider needs to have DNS. DNS records for mailtrap.io need to be set as per this HCA for my hosting provider

<https://help.mailtrap.io/article/128-domain-verification-with-namecheap>

```
# Testing & Dev
MAIL_MAILER=smtp
MAIL_HOST=sandbox.smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=xxxx
MAIL_PASSWORD=xxxxxxxxxx
MAIL_FROM_ADDRESS="noreply@workopia.test"
MAIL_FROM_NAME="workopia.test"
```

```
# Live production
# MAIL_MAILER=smtp
# MAIL_HOST=live.smtp.mailtrap.io
# MAIL_PORT=587
# MAIL_USERNAME=api
# MAIL_PASSWORD=*****xxxx
# MAIL_FROM_ADDRESS="noreply@workopia.test"
# MAIL_FROM_NAME="workopia.test"
```

For a production environment, The live settings need to be changed. It is worth when deploying to live to comment out the dev settings and uncomment the live settings

14. Deploy Using Laravel Forge

[13.1 Intro](#)

[13.2 Prepare & Push to Github](#)

[13.3 Laravel Forge Server & Site setup](#)

[13.4 Domain name Setup](#)

13.5 SSL & launch Test

14.1 Intro



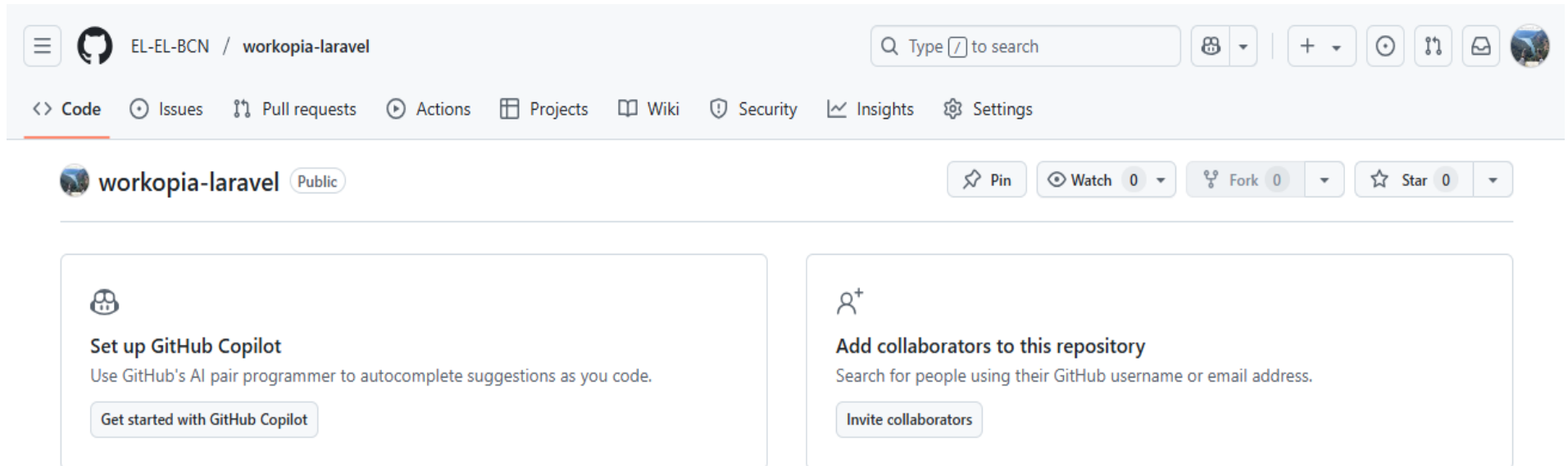
Laravel From Scratch

Deploy Using Laravel Forge - Section Intro

- ✓ **Push Your Code To Github**
- ✓ **Setup Laravel Forge Server**
- ✓ **Link Digital Ocean Account**
- ✓ **Deploy Project**
- ✓ **Add A Domain**
- ✓ **Add a Free SSL**

14.2 Prepare & Push to Github

Ensure that GIT is downloaded on the machine <https://git-scm.com/downloads/win> and create a new repo in github.



Create local repo, add files, initial commit then push to remote repo.

14.3 Laravel Forge Server & Site Setup

Laravel Forge is a server management tool that makes it easy to deploy Laravel applications. It takes care of setting up your environment with everything from PHP to NGINX to your database. It is a paid service, but it is worth it. You can choose the "Hobby" plan for \$12 per month. There is no free solution.

Forge itself does not host your application. It is just a tool that makes it easy to deploy your application to a server. You can use it with any server, but it is most commonly used with Digital Ocean and AWS. I prefer Digital Ocean because it is cheaper and easier to work with in my opinion.

You can use something like Digital Ocean, Linode or AWS on their own without Forge, but that is a lot of work to set up. There is a lot of provisioning that needs to be done and you really need to understand Linux and the terminal. You need to manually install and configure all the software you need and you're almost guaranteed to run into issues unless you're really experienced.

If I deploy a Laravel project from scratch, I would probably follow these commands and not use Laravel forge.

<https://www.namecheap.com/support/knowledgebase/article.aspx/9694/29/how-to-install-laravel-on-our-server/>

<https://dev.to/frankliniwobi/deploy-laravel-projects-to-namecheap-server-on-github-push-with-3-easy-steps-the-only-tutorial-you-would-ever-need-a8h>

Course Content

VScode PHP Intelephense, Laravel Blade Snippets & Laravel Snippets extensions | Laravel herd | virtualbox win10 vm | web.php routing file | adding GET & POST endpoints | CSRF protection | Middleware CSRF exceptions | passing params in URI | param constraints | global constraints | request object | request object query params | query, only, all, has, input keywords | Response Helper | set http code | set content type | set cookie | get cookie | herd secure self signing | create & display views | pass data to view | with method | compact method | blade templates & directives | @if, @else, @foreach, @forelse, @empty, @break, @continue | \$loop->index, \$loop->iteration, \$loop->remaining, \$loop->count, \$loop->even, \$loop->odd | create controllers | using artisan | controller methods | clean web.php routing file | params and requests in the controller | @csrf hidden token | Generate Resource Routes & Methods | type hinting in controllers | Layouts with Template Inheritance | Partial & @include directive | what are components? | components & slots | named slots | z-header, x-slot | install tailwinds css & other dependancies via NPM | Header Component & URL Helper | Conditional Classes | @php Directive | Component Attributes | Props | nav.link & button-link blades using props & attributes | mobile menu nav links | mobile menu toggle | hero component | top and bottom banners | Database options | Create Database & user | Configure Database connection | Migrations Overview & Commands | Creating Migrations | Intro to Models | Fetching Data | Eloquent ORM | Tinker | CRUD Operations | Model Binding | Single Job Listing | Create Job Listing | Input Validation & Errors | Job Schema Update Migration | Eloquent Relationships | Using Factories | Creating Factory & Faker Library | Creating Seeders | Final Database Seeder | Foreign Key errors when seeding to MySQL/MariaDb databases | Job Page | Job Card Component | Homepage Jobs | Job Details Page | Create Job Page | Text Input Component

Other Input components | Finish Input Validation | Flash Messages & Alert Component | Alpine.JS & Alert Dismiss | Optional Job fields | file uploading | dd (die & dump) | symlinks | storage folder | Update Job Listing | Delete Job Listing | Authentication Options | Laravel Breeze Setup | How Sessions Work & Session Helpers | Login & Register Control Routes | Register New user | Log in user | Logout and Auth Directive | Middleware Overview | Protecting Routes (in controller & in routes/web.php with middleware | Guest Middleware | Test User Seeder | Add Current user to Listing | Policies & @can Directive | Policy Authorisation in Controller | Dashboard Controller & View | Dashboard user Job Listings | Profile Controller & Info Update | Profile Avatar upload | Show Avatar in Header | Simple Job Pagination | customising pagination view | Bookmark Migration & Relationships | Seeding Bookmarks | Get & Show Bookmarks | Bookmarking Jobs | Removing Bookmarks | Applicant Migration & Model | Applicant Form Modal with Alpine.js | Fix Modal Blip with x-cloak | Applicant Controller & Store Method | Show Applications to Owner | Delete Applicants | Prevent Multiple Applications | Search Component & Route | Search Functionality | Mapbox Setup | Hide Mapbox key with proxy endpoint | Send Emails with Mailers & Mialtrap | Sending Data in Emails | Email Attachments | Setup Emails For Production | Prepare & Push to GitHub | Laravel Forge Server & Site setup | Domain name Setup | SSL & launch Test